1. Introduction

There is a growing interest in the subject of visual languages, as evidenced by a spate of new publications in the area (including two new books [1,2]) and the birth of a new journal devoted to the subject.

At last year's Vision Interface / Graphics Interface Conference, this author presented a paper on "informatics calculus", a graphical query language for multi-media database systems [3]. "Informatics calculus" has been revised and has been renamed "TableTalk". The syntax of the language has been formalized in terms of syntax diagrams and the semantics has been recast in terms of sequences, objects and their transformations. TableTalk represents an effort to capture all "database activities" in a seamless, visual language. Seamless implies that regardless of the activity, the database user will be operating within the same conceptual and visual framework. Accomplishing this is a major challenge, since database activities are quite diverse. With respect to the writing of applications programs, the fundamental issue for TableTalk is whether a graphical, functional query language can be integrated into a framework which supports applications programming? This paper proposes that this can be accomplished by combining TableTalk with another diagrammatic language, Nassi-Shneiderman (N-S) diagrams.

Experience with visual programming systems such as PICT/D has shown that in order to make a visual language cost effective, the visual expressions in the language must correspond to concepts in a very high level programming language. Graphical query languages generally adhere to this principle.

The term "N-S diagrams" refers to a family of very similar diagrammatic notations whose common idea is to express structured programming control structures in diagrammatic form. N-S diagrams, like TableTalk, express programs as mosaics, that is, as rectangular arrangements of non-overlapping tiles. Spatial patterns are used to denote various kinds of combinators.

N-S diagrams represent a higher level of program abstraction than the icons and flowcharts of PICT/D. N-S diagrams are also far less demanding in terms of computer graphics resources. Thus, they represent a practical approach to facilitating program development. N-S diagrams have been employed by a variety of systems, including IBM's Programming Support System [8], PIGS [9] and GRASE [10]. Figure 1 presents some of the basic tiling patterns employed in N-S diagrams.

2. Nassi-Shneiderman diagrams and graphical query languages

Shu [1] presents an excellent introduction to visual languages. Other introductions to this topic are found in Chang [4] and in Ambler and Burnett [5]. Our specific interest is in visual languages which allow people to program by means of visual expressions. Representative examples of such systems include PICT/D [6], N-S diagrams [7] and all graphical query languages.

Figure 1. Tiling patterns employed by N-S diagrams
Graphical query languages generally operate at a higher level of abstraction than N-S diagrams. Data models have encouraged the development of very powerful, non-procedural, database languages whose fundamental feature is that they describe "programming the large". For example, the relational data model [11] has fostered the development of a great variety of database languages, both textual and graphical. Some of these are based upon relational algebra (e.g., ASTRID [12]), others are based upon relational calculus (e.g., QUEL [13] and QBE [14]) and yet others are designed around slightly different formalisms (e.g., SQL [15]). Shipman's functional data model [16] gave rise to a powerful functional query language, FQL (see Buneman et al. [17]) and the TableTalk language [3], which captures the semantics of a functional query language in diagrammatic terms.

A review of graphical query languages reveals a great diversity in terms of their use of visual information to communicate the underlying database semantics. (See Shu [1], chapters 2 and 11, and also the proceedings of the IEEE Workshop on Visual Languages, including Czejdo, Reddy et al., [18] and Rohr [19].) An interesting discussion of available means for expressing semantics in visual languages is given in Selker and Koved [20]. QBE [14], CUPID [21], ISIS [22], FORMAL [23], and GUL-Q [24] represent a variety of approaches for expressing database semantics in a graphical query language.

There are several fundamental challenges inherent in graphical query language design. These include:

1. Basing the query language on a suitable user-centered data model which can be mapped to the underlying logical model.
2. Employing an effective visual metaphor for expressing database semantics.
3. Achieving orthogonality.
4. Mirroring thought (expressiveness, completeness).
5. Supporting multi-media.
7. Providing a uniform system image across the various database activities (including data definition, querying, updating, report generation and applications programming).

Providing a uniform system image across various database activities (challenge #7, above) is certainly a formidable task. IBM's relational database system DB2 is typical in this regard (see Date [25]). Although IBM supports QBE as a user-oriented, graphical interface to DB2, QBE does not mesh with the applications development language (which is COBOL!). Applications programs are written in COBOL with embedded SQL.

3. TableTalk: A graphical, functional query language

This section presents a brief synopsis of the TableTalk language. TableTalk is based upon a semantic data model. Appendix A presents most of the basic elements of TableTalk's semantic data model and Appendix B shows the semantic database schema assumed throughout the rest of this paper. TableTalk's data model represents a compromise between Hammer and McCleod's original semantic data model [26] and Shipman's functional data model [16].

Here are some essential facts about TableTalk:

1. TableTalk is a functional query language.
2. TableTalk expresses functional programs as mosaics.
3. TableTalk is a sequence processing language.

TableTalk's underlying formalism derives from Buneman's functional query language, FQL [17]. FQL and TableTalk express queries as functional forms which are built up from primitive elements (including entity classes, attributes and constants) using functional combinators (such as composition and Cartesian product). TableTalk uses various visual elements to express functional combinators. For example, functional composition can be expressed by means of vertical juxtaposing of tiles, Cartesian product is expressed by means of horizontal juxtaposing of tiles, and subqueries are expressed using mosaics embedded within tiles. In addition, external functional forms can be referenced by name. In such a situation, the functional form referred to is substituted into the functional form in which the reference occurs. Figure 2 summarizes the basic tiling patterns of TableTalk.

TableTalk is a sequence processing language. This means that the semantics of the language is described in terms of sequences of objects. An object encapsulates one or more pieces of information. There are four kinds of objects: scalar objects, set objects, entity objects and sequences. These correspond to the four kinds of attributes in the underlying model: single-valued scalar, multi-valued scalar, single-valued entity and multi-valued entity attributes. An entity object encapsulates information about some entity in the database. Figure 3 presents graphical representations for the four kinds of objects. (The visual representation of an object is called a "card"). An entity object can be either primordial or non-primordial. For example, a primordial book object encapsulates all of the information associated with a particular book. Since this is potentially an infinite amount of information (due to cyclic patterns in the database schema), the encapsulated information is represented symbolically as an "entity
A non-primordial book object is a book object that has had some of its information filtered out (usually by a process similar to the relational algebraic operation of "projection"). A non-primordial book object can contain other entity objects, either primordial or non-primordial. Figure 4 shows a card for a non-primordial book object.

Figure 2. Basic tiling patterns in TableTalk
Functional composition

Cartesian product

Subqueries

External Reference

Figure 3. Cards for various kinds of objects.
(a) A scalar object

class: REAL
value: 12.345

(b) A set object

class: set of STRING
value: ["PROGRAMMING", "COMPUTER LANGUAGES", "USER INTERFACES", "COMPUTER GRAPHICS"]

(c) A primordial object

class: BOOKS
value: "BC234"

(d) A sequence of primordial objects

sequence class: BOOKS
value: ["BC123", "MR567", "ST123"]

button": A non-primordial book object is a book object that has had some of its information filtered out (usually by a process similar to the relational algebraic operation of "projection"). A non-primordial book object can contain other entity objects, either primordial or non-primordial. Figure 4 shows a card for a non-primordial book object.

Figure 4. Card for a non-primordial book object which contains two scalar objects, a primordial entity object and a sequence of primordial objects.

<table>
<thead>
<tr>
<th>class: BOOKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOK:</td>
</tr>
</tbody>
</table>
| class: STRING
value: "BC234" |
| AUTHOR:      |
| class: AUTHORS
value: "SHU, MAW C." |
| REMARKS:     |
| class: DOCUMENT
value: "" |
| ORDERS:      |
| sequence class: ORDERS
value: ["M789", "G564", "R678", "F678", "T912"] |

Every TableTalk query can be viewed as a sequence of "rows". Each row belongs to a particular syntactical category (determined by the formal syntax of the language, which is given in [27]). Each row can be viewed as transforming an input sequence. It is important to realize that each row describes an iterative process, the iteration being over the objects in the input sequence. The kind of transformation performed by a row depends upon the syntactic category of that row. The major kinds of rows are summarized in Figure 5.

Figure 5. The basic kinds of rows and what they do.

<table>
<thead>
<tr>
<th>row type: entity class / subclass</th>
<th>action performed:</th>
</tr>
</thead>
<tbody>
<tr>
<td>transforms the null sequence into a sequence of primordial objects from the specified class.</td>
<td></td>
</tr>
</tbody>
</table>
| predicate
transformation of any object which does not satisfy the predicate. |
| functional
replacement of a sequence of numbers with a single number. |
| functor
rearranges the input sequence in some way. |
| product
transformation of primordial objects into non-primordial objects by filtering out attributes. |

Graphics Interface '90
3.1 Query sublanguage examples

This section presents a sequence of TableTalk query sublanguage examples.

Example 1. Give the book numbers, author names and photos, for books whose price is less than $10.00.

<table>
<thead>
<tr>
<th>Row</th>
<th>BOOKS</th>
<th>PRICE &lt; 10.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>BOOK#</td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHU, NAM C.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATE, CHRIS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KROENKE, DAVID</td>
</tr>
</tbody>
</table>

In this case, row 1 is an entity class, row 2 is a predicate and row 3 is a product. The result of the query is the sequence of three non-primordial book objects which are displayed in tabular form beneath the double line. The symbol under the PHOTO column represents a "scalar button" which provides access to an author's photo.

Example 2. Give the book numbers, titles, order numbers, and customers for those orders for books published by Addison-Wesley that are out of stock.

<table>
<thead>
<tr>
<th>BOOKS</th>
<th>IN_STOCK = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>&quot;ADDISON-WESLEY&quot;</td>
</tr>
<tr>
<td>BOOK#</td>
<td>TITLE</td>
</tr>
<tr>
<td></td>
<td>ORDERS</td>
</tr>
<tr>
<td>ORDER#</td>
<td>CUSTOMER</td>
</tr>
</tbody>
</table>

The second row of Example 2 is a predicate which is of the form

\[ \text{and} \quad B1 \quad \text{and} \quad B2 \]

where B1 and B2 are boolean expressions. The second boolean expression involves a "derived attribute". A derived attribute is an attribute which is the composition of two or more attributes contained in the database schema. In this case, NAME of PUBLISHER represents the composition of the attributes

\[ \text{PUBLISHER} : \text{BOOKS} \rightarrow \text{PUBLISHERS} \]

and

\[ \text{NAME} : \text{PUBLISHERS} \rightarrow \text{STRING} \]

TableTalk uses the reserved word \# to denote functional composition on the left and the reserved word -\# to denote functional composition on the right. Hence, NAME of PUBLISHER and PUBLISHER's NAME denote the same derived attribute.

Example 3. Give the average price of books written by Chris Date and published by Addison-Wesley.

<table>
<thead>
<tr>
<th>BOOKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>&quot;DATE, CHRIS&quot;</td>
</tr>
<tr>
<td>PUBLISHER</td>
</tr>
<tr>
<td>&quot;ADDISON-WESLEY&quot;</td>
</tr>
<tr>
<td>PRICE</td>
</tr>
<tr>
<td>average</td>
</tr>
</tbody>
</table>

Example 4. Give the title, price and number in stock of all books either written by Chris Date or published by Addison Wesely and give these in alphabetical order by title.

<table>
<thead>
<tr>
<th>BOOKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHOR</td>
</tr>
<tr>
<td>&quot;DATE, CHRIS&quot;</td>
</tr>
<tr>
<td>PUBLISHER</td>
</tr>
<tr>
<td>&quot;ADDISON-WESLEY&quot;</td>
</tr>
<tr>
<td>PRICE</td>
</tr>
<tr>
<td>avg</td>
</tr>
<tr>
<td>TITLE</td>
</tr>
<tr>
<td>sort by</td>
</tr>
<tr>
<td>TITLE</td>
</tr>
</tbody>
</table>

Example 5. Give the book numbers, titles and author's names for all books not written by Chris Date but published by Addison-Wesley that are more expensive than the cheapest book written by Chris Date.

<table>
<thead>
<tr>
<th>BOOKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUBLISHER</td>
</tr>
<tr>
<td>&quot;ADDISON-WESLEY&quot;</td>
</tr>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>of AUTHOR</td>
</tr>
<tr>
<td>&quot;DATE, CHRIS&quot;</td>
</tr>
<tr>
<td>PRICE</td>
</tr>
<tr>
<td>&gt; minimum</td>
</tr>
</tbody>
</table>

The second row of Example 5 is a predicate of the form B1 and B2 and B3, where Bj, j = 1, 2, 3 are boolean expressions. The third boolean expression includes a subquery. This subquery returns a single scalar object, whose value is the lowest price charged for any book written by Chris date. In the third row (a product), the derived attribute AUTHOR's NAME could also have been expressed as:

\[ \text{AUTHOR} \rightarrow \text{NAME} \]
Example 6 illustrates the recursive syntax and semantics of the TableTalk language. The multi-valued entity attribute ORDERS yields, for each book, a sequence of order objects. TableTalk treats all sequences, whether at the top level (i.e., the main processing sequence) or at a lower level (e.g., a sequence within an object), in a consistent manner. Example 6 applies a predicate and a functor (sort by) to the sequence of orders contained within each book object in the main processing sequence.

3.2 Update sublanguage examples

The general form for a database update is shown below:

\[
\text{<ENTITY CLASS or SUBCLASS>} \\
\text{<PREDICATE>} \\
\text{<UPDATE OPERATOR>: <OPERAND(S)>}
\]

The update operators include assign, input, delete, and insert. Each operator has its own set of legal operands.

Example 7.

Increase the price of all books published by Addison-Wesley and written by Chris Date which discusses database systems by 25%.

The second boolean expression in the predicate row of Example 8 includes a subquery which begins with the reserved word self. This subquery is re-computed for each individual customer object in the main processing sequence. The value of the subquery is the number of orders for the current customer.

Example 8. Delete all customers who live in Virginia who have placed fewer than two orders.

4. Integrating TableTalk and N-S diagrams

TableTalk and N-S diagrams share the use of mosaics as a means of representing programs. A TableTalk mosaic represents a program in a functional query language and an N-S diagram represents a program in an imperative language. We propose a melding of TableTalk mosaics and N-S diagrams as a means of achieving a multi-paradigmatic, diagrammatic language for specifying database applications. The tiling metaphor can act as a bridge between the functional programming paradigm of TableTalk and the imperative paradigm of N-S diagrams.

Integrating TableTalk and N-S diagrams involves solving three fundamental problems:

1. Incorporating N-S notation into the TableTalk language.
2. Incorporating TableTalk into the N-S language.
3. Providing a mechanism for the functional language to communicate with the imperative language, and visa versa.

Problem 1. The TableTalk syntax needs to be extended to include N-S diagrammatic notations. Example 11 shows an N-S diagram which includes an embedded TableTalk update. The embedded TableTalk update is bounded on the left by a shaded
Problem 2. TableTalk is a sequence processing language. Iteration over objects in a sequence is automatic. Some device is needed to allow the imperative language to express iterations over TableTalk sequences. In embedded SQL this is accomplished by means of cursors. Cursors allow an imperative language to process the results of a query record by record. This enables fine-grained interactions between the imperative language and the database language. A similar device will be necessary in the final specification of the integrated TableTalk / N-S language.

5. Summary and conclusions
This paper presented Tabletalk as a database language and did not discuss the implementation of the user-interface. Tabletalk is intended to be implemented as a highly interactive system with extensive help facilities. In particular, the user will not need to have an in-depth knowledge of the database schema. Randy Kirsh at Temple University is currently implementing a small subset of TableTalk in order to explore the relationship between such a schema-based help facility and the language structure.

Visual languages, in order to be cost effective, must allow programmers to work at a high level of abstraction. Unfortunately, most languages which operate on a very high level of abstraction, such as database query languages, lack flexibility and generality. Consequently, there is little commonality between end-user interfaces and the languages which are used to develop applications.

This paper points out the similarities between TableTalk, a functional, graphical query language, and N-S diagrams, a family of diagrammatic languages for specifying control structures in imperative languages. The paper concludes by presenting some tentative ideas concerning how TableTalk and N-S diagrams can be integrated in order to yield a diagrammatic applications programming language.

5. References

Graphics Interface '90


APPENDIX A

BASIC KINDS OF OBJECTS IN TABLETALK'S SEMANTIC DATA MODEL

<table>
<thead>
<tr>
<th>kind of object</th>
<th>example of this kind of object given in Appendix B</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY CLASS</td>
<td>BOOKS</td>
</tr>
<tr>
<td>SUBCLASS</td>
<td>BEST_SELLERS</td>
</tr>
<tr>
<td>SCALAR CLASS</td>
<td>STRING</td>
</tr>
<tr>
<td>ENTITY ATTRIBUTE</td>
<td>the (multi-valued) entity attribute: ORDERS: BOOKS ----&gt; ORDERS</td>
</tr>
<tr>
<td>SCALAR ATTRIBUTE</td>
<td>the (single-valued) scalar attribute: BOOKS: BOOKS ----&gt; STRING</td>
</tr>
<tr>
<td>DERIVED ATTRIBUTE</td>
<td>the attribute: NAME of AUTHOR: BOOKS ----&gt; STRING</td>
</tr>
<tr>
<td>INVERSE ATTRIBUTES</td>
<td>the attributes: ORDERS: BOOKS ----&gt; ORDERS</td>
</tr>
<tr>
<td>INTERSECTION ATTRIBUTE</td>
<td>the attribute: QUANTITY: (BOOKS p ORDERS) ----&gt; NUMBER</td>
</tr>
</tbody>
</table>

APPENDIX B

SCHEMA DIAGRAMS ASSUMED BY TABLETALK EXAMPLES IN THIS PAPER

This appendix presents the database schema which is assumed in all examples given in this paper. Figure B.1 shows the database schema in terms of entity classes, subclasses and their relationships. Figure B.2 shows individual entity classes and subclasses and their scalar attributes. Entity classes and subclasses are denoted using boxes. Scalar classes are denoted using ovals. Multi-valued attributes are denoted using double-headed arrows and single-valued attributes are denoted using single-headed arrows. Inverse attributes are connected by a single line, which might have an intersection attribute emanating from it. QUANTITY is a single-valued scalar intersection attribute.
Figure B.1

The database schema in terms of entity classes, subclasses and their relationships. Also shown is the intersection attribute, QUANTITY.

Figure B.2 Entity classes and subclasses with their scalar attributes.

(a) CUSTOMERS

(b) ORDERS

(c) BOOKS

(d) BEST_SELLERS

* Address has been simplified. It actually consists of a collection of attributes, namely, STREET, CITY, STATE and ZIP_CODE.