Multi-Media Database Language SL/B5 based on Screen Flow

Fumio KITAGAWA and Hideto IKEDA

Information Processing Center, Hiroshima University
1-1-89, Higashisenda, Naka, Hiroshima 730 JAPAN

Abstract

A very significant goal for multi-media database systems (MMDBS) is the integrated management of unstructured data such as drawings, images, text and voice as well as formatted or structured data. So it is required for MMDBS to harmonize both multi-media data storage and processing capabilities. Traditional technology for data management is based on storage and retrieval of structured data of numbers and characters. But the current technological environment is not sufficient to implement practical application systems in the storage and processing of unstructured data. In order to overcome the problems involved in implementing MMDBS, this paper proposes a new multi-media data processing language SL/B5 based on screen flow. A program of SL/B5 is represented by a collection of screens which are connected through what we have termed "screen call". The specifications of each screen may be divided into four parts, IMAGE for formatting multi-media data on a display terminal with speakers, FUNCTION for data entries and transformations, DATABASE for database accesses and CALL for conditional call of other screens with parameters. The database is a collection of 5-tuple (value, data type, assigned with a key, entity, term of validity) which can include multi-media objects, meta-data, user information, security and integrity specification. The special database structure is called B5 structure. SL/B5 provides us powerful, flexible and friendly multi-media data processing capabilities based on flexible database structure B5. SL/B5 was specially designed to support development of information systems by naive end-users in considering the harmonization to B5.

KEYWORDS: multi-media database system, screen language, screen flow control, multi-media data storage B5, end user computing, 4th generation language.

1. Introduction

The majority of efforts in database research have been spent in developing database management systems (DBMSs) for traditional business applications, such as employer administration and inventory control that use well formattable or structurable data. The success of DBMS architecture is derived from various requirements that cannot be sufficiently satisfied in traditional DBMSs.

Some DBMSs were expanded to meet a part of the requirements from nontraditional applications, e.g., engineering database systems, image database systems, statistical database systems and knowledge-base systems. These expansions are basically made in two ways, i.e., data types and data manipulation capabilities. Improvement of performance and user interface are involved in the two expansions.

Some other DBMSs were expanded as total tools for developing and maintaining information systems by integrating application packages, report writers, query languages, statistical packages, data dictionary systems and end-user-oriented programming languages. Among these integrated software, some were improved for productivity of system development and were called fourth generation languages.

The two different trends of DBMSs suggest that a significant problem for future DBMSs is the harmonized integration of storage and processing for multi-media data. From these considerations, future DBMSs must satisfy the following requirements

(1) Meaningful integration of multi-media data
There are many requirements in multi-media database management (e.g. Christodoulakis et al 84, Stonebraker and Rowe 85). Future multi-media DBMSs should support nontraditional data types, complex data types and abstract data types. Multi-media data is created from some other data by compositions. For example, a driver's license is a multi-media data which is composed of a photo and signature (image data) and some descriptions, ID number, name, data of issued type, class, data of expiry (character data) and so on. Although traditional DBMSs have tried to manage some relationships of data with integrity, they were not sufficient for nontraditional applications. Especially, hierarchical relationships by data abstraction. It is very important to support such hierarchical relationships of data because meaningful integration of various data types is one of the final goals for multi-media DBMSs.

(2) Integration of preexisting applications
For future DBMSs, it is not a common recognized require-
ment but it is very important to continue the support to preexisting applications, and, if possible, to solve some practical problems of them; e.g., productivity, flexibility, maintainability, performance, reliability, usability of data and user-friendliness. Users want to use one DBMS to implement nontraditional application systems and to transfer their preexisting applications without changing programs and data.

2. Multi-media Storage

2.1 Structure of B5

We propose a new database storage B5 in this section. B5 is a (large) sheet which has five columns, (VALUE, TYPE, ATTRIBUTE, OBJECT, TERM-OF-VALIDITY).

Our database is a collection of such 5-tuple, which is called a fact. An object to be managed in the system is identified by object name. A property of observed information of an object is distinguished by an attribute value. In order to store a fact of an object, it is necessary to express it as a manageable value in the computer, usually a bit string. We call it object value. There are various ways to express a fact, e.g., fixed point binary, floating point binary, packed decimal, unpacked numeric. If we treat the value meaningfully, we need description of the data type and term of validity for the fact. For example, a fact of B5 (H. IKEDA, CHAR(8), NAME, STAFF, 1988.4.1-*) means that character string "H. IKEDA" is a name of a staff member from April, lst, 1988.

As a value, we can accept complex expressions of other objects be setting and listing. The previous example driver's license can be stored as a set of facts;

( I230-3204-8092, *, LICENSE#, 301332145, 10-31-87 to 03-30-91 )

( ORIGINAL, *, TYPE, I230 3204-8092, * )

( NONE, *, RESTRICTION, I230-3204-8092, * )

( LICENSE#, CHAR(8), ID, LICENSE, *-* )

( CHAR(<= 10), CHAR(7), DATA-TYPE, LICENSE#, CHAR(14), DATA-TYPE, LICENSE#, CHAR(<= 10), CHAR(7), DATA-TYPE, LICENSE# )

( CHAR(<= 20), CHAR(10), DATA-TYPE, RESTRICTION, *-* )

( IMAGE(200,100,M), CHAR(16), DATA-TYPE, SIGNATURE, *-* )

( IMAGE(200,600,C), CHAR(16), DATA-TYPE, PHOTO, *-* )
The later facts correspond to meta-data in traditional DBMSs. Such meta-data are also treated in B5. Attribute name ID has special meaning in B5, i.e., the corresponding value (301332145) is used as an identification of the object (PERSON). Such names are referred to "reserved words", that have special meaning and are attached special functions in B5. DATA-TYPE, NAME, OBJECT, ATTRIBUTE and TERM-OF-VALIDITY are also reserved words. The other reserved words include SYSTEM, ELEMENT, OCCURRENCE, FUNCTION, RELATION, FORMULA, USER, CONSTRAINT, EXPRESSION and so on.

2.2 Database Access in B5
The most simple query is a 5-tuple that is created from a fact by replacing some elements with variables. In the license example, the following tuples are queries:

( X, +, LICENSE#, 301332145, * )
( Y, +, PILOT#, 301332145, 12-01-88 )

which have the meaning of:

What is the license number of driver #301332145?
Show me a photo of driver #301332145 at December 1st, 1988.

respectively.

Complex queries are available in B5, such as:

\[ \exists Y \ ( X, \ast, \text{name}, Y, \ast ) \]
\[ \& ( Y, \ast, \text{ID}, \text{person}, \ast ) \]
\[ \& ( A, \ast, \text{CLASS}, Y, \ast ) \]

which means that "List up all names of persons who have a driver's license of class A". The answer to this query is a set of names because the number of persons who satisfy the conditions is possibly two or more. A variable with quantifier, such as Y, is called a bounded variable and a variable without quantifier, like X in the above query is called unbounded. A 5-tuple which includes unbounded variables can be said to be a query. Symbol "\&" means "and", which is one of the logical operators. The other logical operators are "or" and "not". The other quantifier than "\exists" is "\forall" in B5. A general query is composed with some simple queries, quantifiers, and logical operators.

Users can specify a query in a more simple way, e.g.,

( X, +, NAME, PERSON(Y), \ast )
\& ( A, \ast, CLASS, Y, \ast )

for the same query above because the reserved word ID plays a special role for such retrieval.

3. Screen Language SL/B5
3.1 General Concept of Screen Language SL/B5
SL/B5 is a programming language which is supported in multi-media storage B5. Traditional procedural languages focus to describe behavior of the target program in the computer. Generally, a program written in a procedural language, PL/1, COBOL and PASCAL is a sequence of instructions which are executable in the target computer.

A program written by SL/B5 is a collection of screens, which can be shown in the target display terminal with speakers. A screen is identified by a name and consists of four sections, i.e., IMAGE, FUNCTION, DATABASE, CALL. Figure 1 illustrates a screen.

A screen is a unit of execution of a program like a sentence of PL/1 or PASCAL. But the function is quite different from a sentence. PL/1 supports various sentences which have different functions. SL/B5 supports execution of screens which provide the same type of capabilities, that is, formatting data to be shown on the target display terminal with speakers, data transformations, database access and transfer the control to another screen. Namely, a program of traditional programming languages is a sequence of heterogeneous instructions, input/output sentences, data transformation sentences and control sentences. On the other hand, a program of SL/B5 is a collection of screens which have homogeneous functions. This is the most different feature of SL/B5. This difference is similar to that between "Blocks" and "Legos". "Blocks" are a set of pieces with different shapes, balls, rectangular plates, triangular plates, sticks and so on. "Legos" are a collection of same pieces. However both toys are for construction. As similar as a complex (toy) building can be constructed by some simple legos with the same shape, a complex program can be described by some simple screens with same type.

The other significant difference of SL/B5 from traditional programming languages is to create a program from external behavior of the target system. This approach is usually used in prototyping or requirement specification languages, and is advantageous in comparison to traditional ones.

3.2 Structure of B5-Screen
A screen of SL/B5 consists of four sections, i.e., IMAGE, FUNCTION, DATABASE and CALL. Appendix I defines the syntax of screen. In this section, we shall describe some semantics of each section.

The IMAGE Section specifies how to create a screen on a bit-mapped display with speakers. All data other than voice can be displayed on a bit-mapped screen. The major feature of IMAGE Section is to map displayable data as image data on a target screen. The Background can be modified for effective representation of data.

The FUNCTION Section plays the role of data transformations, e.g., arithmetic computations, format/data type transformations, image processing and even data entry from the keyboard, mouse, touch screen and microphone. Basically, all data must be transformed to image data or voice in this section. But a user can eliminate the last transformation to image data, only when it is natural and trivial one.

The DATABASE Section includes all updates of the database B5. We can specify them in a very simple way, because the database is organized in a B5 format. Conditional updates are also available by using IF clauses.

The CALL Section can be used for realizing a screen flow. IF a condition specified in IF clause of this section is satisfied, the control of execution is transferred to a corresponding screen with parameters.
DEFINE SCREEN study_of_flower 1024 x 760;

IMAGE
  X1 AT (0, 0) – (1023, 759);
  "STUDY OF FLOWER" AT (100, 100) – (880, 150) WITH SIZE(24), BASE(GREEN), BOX(SINGLE);
  X2("EXAMPLE") AT (40, 600) – (400, 650) WITH SIZE(24), BASE(BLUE), BOX(SINGLE);
  X3("STRUCTURE") AT (40, 700) – (480, 750) WITH SIZE(24), BASE(BLUE), BOX(SINGLE);
  X4("FAMILY") AT (600, 600) – (820, 650) WITH SIZE(24), BASE(BLUE), BOX(SINGLE);
  X5("END") AT (600, 700) – (770, 750) WITH SIZE(24), BASE(RED), BOX(SINGLE);

FUNCTION
  X1 = SELECT(*, "picture", "blue capensis", *);
  X6 = TRANS("Select one subject by touching the screen", A), TYPE(FEMALE);

DATABASE
  SCR-NAME AS "history_of_study" OF USER-ID AT B5-TIME;

CALL
  example_of_flower IF INPUT(TOUCH).IN. X2;
  structure_of_flower IF INPUT(TOUCH).IN. X3;
  family_of_flower IF INPUT(TOUCH).IN. X4;
  EXIT IF (INPUT(TOUCH).IN. X5).OR. TIMEOUT(5 min);

FIGURE 1-a An example of screen definition

FIGURE 1-b Behavior of the screen defined as Figure 1-a
3.3 Special Features of SL/B5

In this section, we shall show some special features of SL/B5 as follows:

(1) **Higher modularity of information** is the most significant design policy of SL/B5. As we stated above, a program of SL/B5 is a collection of homogeneous screens. These characteristics derive from easy understanding and ease-of-use of information resources. It also makes it easy to add a new resource by modifying some pre-defined resources. Another effort for higher usability of screens is recursive call capability. Figure 2 is an example of query screen, which asks the name of a flower on the display. When a flower is chosen to one of another kind, the screen can be available because this screen is not dependent on a particular kind of flower. Figure 2 is also an example of recursive call capability. By parameterizing a name of a flower, we can use the same screen for both query and answer.

(2) **Audio-visualization of all information** is also a basic policy for SL/B5 design. All data can be mapped to a movie, that is, a series of image screens with synchronized voice. A traditional way how to store information in the computer is a coding representation. If we want to see the information, we must transform it to image data on a screen, paper, plotter or voice on a speaker. Characters are displayed as a font on a screen. A function can also be seen as a graph, that is to say a kind of image data. Therefore, true transformation of data is a mapping to image data. In the FUNCTION Section of SL/B5, all data to be displayed on a screen, which are also included in the IMAGE Section, must be mapped to image data. Figure 3 illustrates of conceptual diagram for data transformation in SL/B5.

(3) **Orthogonalization of functions** is important for design of every language. Especially, a multi-media DBMS requires many functions of data transformation, because it is necessary to support individual functions for each data type and data transformation for each other. If we would support simple union of individual functions for each data type, the number of all functions in the system would be very vast and we could not learn and use it. An approach to solve this problem is the orthogonalization of functions, that is, using the same name for functions having similar capabilities and different data types. It, however, is easier to preach than to practice this. Appendix II is the list of functions to be supported in SL/B5. The design policy was orthogonalization of functions.

4. Discussion and Conclusion

This paper proposed a new multi-media data processing language SL/B5. The special characteristics are:

(a) Screen-based language
(b) Multi-media database storage B5
(c) Orthogonal set of functions
(d) Audio-visual representation

On the other hand, we have discussed requirements that future DBMSs should satisfy in the Introduction. Now we shall check each requirement that can be attained by SL/B5. Meaningful integration of multi-media data can be implemented by the special multi-media storage B5 and functions given in SL/B5. Although we could not sufficiently present a detail of B5, it has enough capabilities for the integration. We believe that SL/B5 can support basically almost all preexisting applications of traditional DBMSs. But we will check the functional capabilities and performance by prototyping and carry out minor change if necessary. This requirement is a requirement which we cannot neglect from a practical point of view. Screen interface and multi-media support of SL/B5 will improve user interface and productivity of programs by end-users.

This language SL/B5 and database storage B5 are now developing a prototype system by the support from Tokyo Electric Company (TEPCO) in Japan. The first application of SL/B5 will be a CAI system of Hiroshima University.

References

When this screen is called with a parameter, the behavior of this screen is different from itself called with another parameter (Figure 2-b, 2-c).

**FIGURE 2-a** An Example of definition for multi behavior screen

**FIGURE 2-b** One behavior of the screen called by 'CALL question_of_flower("lotus")'

**FIGURE 2-c** Another behavior of the screen called by 'CALL question_of_flower("orchid")'
### Appendix I  SYNTAX OF SL/B5

In the following description, a word enclosed "[" and "]" is omittable.

- **object** := numeric.character | character | sound.element | pixel | variable
- **function** := character.string
- **variable** := &character.string
- **object.type** := N | C | A | I | L | S | D
- **screen** := DEFINE SCREEN
  - **screen.name** (parameters; recursive.parameters) [size];
  - **IMAGE.** Section
  - **FUNCTION.** Section
  - **DATABASE.** Section
  - **CALL.** Section
- **screen.name** := character.string
- **parameters** := image.id | variable
- **size** := h.size x v.size
- **image.id** := Xnumeric.characters
- **h.size** , **v.size** : positive integer

### FIGURE 3  Conceptual diagram for data transfer

<table>
<thead>
<tr>
<th>form of object</th>
<th>additional attributes by data transfer</th>
<th>specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>code-data</td>
<td>type {font, tone}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>effect {color, decoration, music-interval, tempo}</td>
<td></td>
</tr>
<tr>
<td>image-data</td>
<td>arrangement {position, time}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>font-size</td>
<td>IMAGE Section</td>
</tr>
<tr>
<td></td>
<td>impose {replace, OR, AND, XOR}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>box of area</td>
<td></td>
</tr>
<tr>
<td></td>
<td>base color of area</td>
<td></td>
</tr>
<tr>
<td>screen</td>
<td>device number</td>
<td>manual operation</td>
</tr>
<tr>
<td></td>
<td>specify some windows</td>
<td></td>
</tr>
<tr>
<td></td>
<td>effect {volume, surround, echo}</td>
<td></td>
</tr>
<tr>
<td>device</td>
<td></td>
<td></td>
</tr>
<tr>
<td>person</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
image.description_string := image.description
                | image.description_string ; image.description
image.description := image.id [( object )] image.environment
                | object image.environment
image.environment := AT area [ WITH d.attributes ] [ WHEN period ]
                | WHEN period
area := address[-address] | variable
d.address := ( integer, integer )
d.attributes := display.attribute
                | d.attributes, display.attribute
display.attribute := SIZE ( s.parm ) | IMPOSE ( i.parm ) | BASE ( c.parm ) | BOX ( b.parm )
s.parm := integer
i.parm := REP | AND | OR | XOR
c.parm := WHITE | BLACK | RED | BLUE | GREEN | YELLOW | PURPLE
                | PINK | BROWN | LIGHT.BLUE | LIGHT.GREEN
                | LIGHT.YELLOW | THRU
b.parm := NONE | SINGLE | DOUBLE | BOLD
period := time, time
time := integer min integer sec
                | integer min
                | integer sec

FUNCTION.Section
FUNCTION.Section := FUNCTION transform.descriptions;
transform.descriptions := transform.formula [ IF condition.formula ]
                | transform.descriptions ; transform.formula [ IF condition.formula ]
transform.formula := variable = object [ transforms.attributes ]
transform.attributes := [ TYPE ( t.parm ) ] [ EFFECT ( e.parms ) ]
t.parm := ITALIC | GOTHIC | MINCHO
                | MALE | FEMALE | CHORUS | PIANO | VIOLIN | FLUTE
                | BIRD | BELL | BUZZER | CHIME
e.parms := e.parm | e.parms, e.parm
e.parm := c.parm
                | UNDERBAR | SHADOW | SLANT | BOLD | EDGE | WIDE | SLIM
                | OCTAVE | ( integer ) | TEMPO | ( integer )
condition.formula := object . relation . object | attention
                | ( \exists variable ) condition.formula
                | ( \forall variable ) condition.formula
                | condition.formula . AND . condition.formula
                | condition.formula . OR . condition.formula
                | NOT. condition.formula
                | ( condition.formula )
relation := EQ | GE | GT | LE | NE | INC | ELM | IN
attention := ATTEN | TIMEOUT ( time ) | ERROR

DATABASE.Section
DATABASE.Section := DATABASE update.description.string;
update.description.string := update.description
                | update.description.string ; update.description
update.description := value AS attribute OF entity AT term [ IF condition.formula ]
                | ( value, attribute, entity, term : object )

CALL.Section
CALL.Section := CALL screen.call.descriptions;
screen.call.descriptions := screen.name ( put.parms ; put.r.parms ) [ IF condition.formula ]
                | screen.call.descriptions ; screen.name ( put.parms ; put.r.parms ) [ IF condition.formula ]
                | ( put.parms, put.r.parms : object )
**Appendix II  THE LIST OF FUNCTIONS**

**TABLE 1. Functions**

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>object</th>
<th>role of function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>object, num, condition</td>
<td>Select num-elements from the object by condition. (num : integer)</td>
</tr>
<tr>
<td>APPEND</td>
<td>object1, object2, num</td>
<td>Append object2 to object1. When two objects are list-type, num indicates the order of insert position.</td>
</tr>
<tr>
<td>DELETE</td>
<td>object, condition</td>
<td>Delete elements by condition.</td>
</tr>
<tr>
<td>CHANGE</td>
<td>object1, object2, condition</td>
<td>Change a subset/sublist of object1 that is selected by the condition into object2.</td>
</tr>
<tr>
<td>*</td>
<td>object1, object2</td>
<td>Product of object1 by object2</td>
</tr>
<tr>
<td>+</td>
<td>object1, object2</td>
<td>Sum of object1 and object2</td>
</tr>
<tr>
<td>-</td>
<td>object1, object2</td>
<td>Difference between object1 and object2</td>
</tr>
<tr>
<td>/ COUNT</td>
<td>object1, object2</td>
<td>Quotient (object1 divided by object2)</td>
</tr>
<tr>
<td>INDEX</td>
<td>object1, object2</td>
<td>The number of elements of the object</td>
</tr>
<tr>
<td>TRANS</td>
<td>object, parameters</td>
<td>Get the position of object2 in object1.</td>
</tr>
<tr>
<td>EDIT</td>
<td>object, parameters</td>
<td>Transform the object by the parameters.</td>
</tr>
<tr>
<td>INPUT</td>
<td>device-name, range</td>
<td>Edit the object by the parameters.</td>
</tr>
<tr>
<td>O-TYPE</td>
<td>object</td>
<td>Input data from device.</td>
</tr>
</tbody>
</table>

**TABLE 2. Functions to get a system value**

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>role of function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>GET year, month, date</td>
</tr>
<tr>
<td>TIME</td>
<td>GET hour, minute, second</td>
</tr>
<tr>
<td>B5-TIME</td>
<td>GET DATE and TIME with B5 format</td>
</tr>
<tr>
<td>USER-ID</td>
<td>GET USER-ID</td>
</tr>
<tr>
<td>SCR-NAME</td>
<td>GET screen-name in which this function specified</td>
</tr>
<tr>
<td>SYS-NAME</td>
<td>GET system-name</td>
</tr>
</tbody>
</table>

**Appendix III  SPECIAL SCREENS**

<table>
<thead>
<tr>
<th>SCREEN</th>
<th>role of screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT</td>
<td>EXIT from SCREEN to OS</td>
</tr>
<tr>
<td>PAUSE(\text{next-screen(parms)})</td>
<td>PAUSE until to get a reply</td>
</tr>
<tr>
<td>SHELL(\text{next-screen(parms), command})</td>
<td>execute OS command</td>
</tr>
<tr>
<td>RETURN(parm)</td>
<td>RETURN to called screen</td>
</tr>
</tbody>
</table>