PALET: A Flexible Office Form Management System

KUNITOSHI TSURUOKA*, KAZUO WATABE*,
and YOSHIYUKI NISHIHARA*

Outstanding features of an office form management system, called PALET, are described. The system designs, processes and manages forms or tables for distributed personal or sectional office applications. PALET users are able to create and change the form structure dynamically on the screen. This creation or change causes the automatic generation or update of the logical and physical data structure. The dynamic form structure change (in non-first-normal-form) is possible without database reconstruction. Other features, such as non-first-normal-form queries, derived form management, and remote query/mail functions, are useful for performing office applications easily and speedily. For unintegrated miscellaneous office applications (not based on integrated databases), PALET enables office workers to flexibly handle various changeable office forms with little knowledge of programming, and of preparation or maintenance of databases.

1. Introduction

Office workers use various business forms in their daily work. Some of these forms may be handled by a total application system, such as a corporate sales management system. However, there are a lot of forms that are only the concern of a specific person or in a specific department. For these miscellaneous forms, it may not be practical to define databases and to write application programs for each form. The reasons are: there are many different types of forms, there is only a small quantity of each type, the data processing is simple, the life of a form is short, and the format is likely to change from time to time.

Some form management systems [1,3,4,5,8,9,10] allow form query and form procedure execution. However, since they aim at integrated database application systems, they need relatively difficult database definition as pre-processing (except[5]). Moreover, they can not flexibly manage the data structure change (if reconstruction is needed, for example). Such database preparation and maintenance is a heavy burden to office workers.

PALET does not aim at the "integrated" system built on carefully designed databases. The PALET objective is to give office workers a very flexible and ready-to-use interface to design and handle unintegrated miscellaneous forms. A PALET user handles forms on the screen, and the on-screen format is directly connected with the form logical data structure as well as with its physical (file) structure. PALET automatically generates these logical and physical data structure from the screen format. One of the outstanding PALET features is its flexibility for change. Users can change the form logical structure, through the screen format, dynamically during program execution. This is possible even after form data have been stored, without database reconstruction (see 3.4). Additionally, PALET has some important features, such as non-first-normal-form queries, derived form management, and remote query/mail functions.

2. Form

This section gives basic concepts and terms for PALET. A form is a structured object, which consists of tables. A form, unlike a text document, has a given data structure and is considered as a unit for data processing purposes. Fig. 1 shows an example.

A form consists of several frames. There is one special frame, called a title, in each form. A normal frame is a two-dimensional table, which can contain items and group items. An item or a group item has a name and value (s), and the value(s) may be repeating. A frame itself is considered to be a group item, and has a name, Frame* (* is a number). An item is called a derived item, if its value is derived from other items. An item name or item value is expressed in a field on the screen. A field may be rectangular (having multiple lines).

A form has a type (which may correspond to a relation) and several instances (which may correspond to tuples). A form type can be seen as a non-first-normal-form relation[6] (multi-level nesting is not allowed, however). An item that identifies the instance is called a primary key. PALET manages a form on three levels--external, logical, and physical. There is an external form, a logical form, and a physical form, as shown in Fig. 2. An external form is the representation on the screen or on paper. A logical form is structured data in a memory, which can be processed by application programs. A physical form is the internal representation in

*C&C Systems Research Laboratories, NEC Corporation, 4-1-1, Miyazaki, Miyamae-ku, Kawasaki, 213, Japan

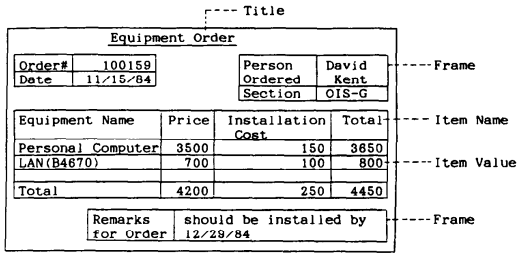

Fig. 1 A Form

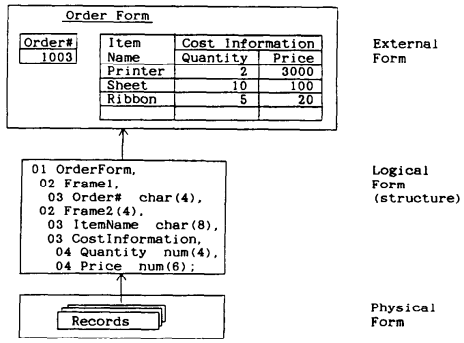

Fig. 2 Form Hierarchy

the secondary storage.

A form schema is a form description which defines a form type. Since a form description consists of three parts, corresponding to each level (as in Fig. 2), a form schema has three components--an external form schema, a logical form schema, and a physical form schema.

A set of form types is contained in a document base. Several document bases are contained in an office. An office is a site where a number of form applications are carried out. An office contains a number of users. A user makes use of several document bases, each corresponding to several applications. PALET manages a set of form types existing in several distributed offices. Actually, PALET does not use any database. This is because current databases have little flexibility for meeting a requirement for data structure change, and are not well suited for unintegrated miscellaneous forms. Although PALET can also handle text documents, attention is mainly focused on forms hereafter.

3. Data Definition

3.1 External Form Definition

An office worker can design an external form type on the screen interactively. Normal designing steps are as shown in Fig. 3. First, the user creates several frames on the screen, as in (a). Next, he divides them to make tables, and expands or reduces rows/columns to desired

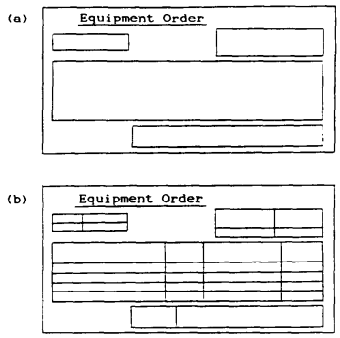

Fig. 3 External Form Design

size, resulting in (b). Finally, he inputs item names into appropriate fields. Important operations are listed below.

(1) Frame--create, divide, delete, move, copy

(2) Row/Column--add, delete, expand, reduce, exchange

(3) Table Head/Side--expand, divide, merge

This operation creates a group item, as shown in Fig. 2.

(4) All Frames--delete, copy (except title)

(5) Item Name--add, delete, modify, copy

An important feature is that each operation can be executed even if the form type already has some form instances (see 3.4). Dynamic form schema change is possible during program execution. The external form design generates an external form schema. The external form schema contains such information as item names, positions and rectangle sizes of item names/values, line information, repeating directions for repeating items, and numeric editing information.

3.2 Logical Form Definition

An important PALET point is that the logical form definition is automatically derived through the external form definition process. That is, the basic part of a logical form schema is generated from an external form schema automatically. (Complete logical form descriptions can not be obtained from the screen format. As described later, information such as item generation rules, some data types, and relationship definitions has to be given by users.) Users do not have to define the database data description before the form processing. This design scheme is the reverse of that for normal integrated database systems. A PALET user designs an external form on the screen, and that description is used by the system to generate a part of the document base definition. PALET can generate a fairly complex logical form in non-first-normal-form, such as the one in Fig. 4. More importantly, the logical form structure can be flexibly changed (see 3.4).

Figure 4 explains the logical form schema generation. When a user draws an external form schema, the system

External Form Schema

Equipment Order

Order#		Person Ordered	
Date		Section	

Equipment Name	Price	Installation Cost	Total
Total			

Remarks for Order	

Logical Form Schema

```
01 EquipmentOrder,
 02 Frame1,
  03 Order# char(8),
  03 Date char(8),
 02 Frame2,
  03 PersonOrdered char(20),
  03 Section char(6),
 02 Frame3,
  03 Group3G(3),
   04 EquipmentName char(20),
   04 Price num(4),
   04 InstallationCost num(4),
   04 Total num(6),
  03 TotalPrice num(6),
  03 TotalInstallationCost num(6),
  03 TotalTotal num(6),
 02 Frame4,
  03 RemarksforOrder char(30);
```

Logical Form Type

Order#	Date	Person Ordered	Section	Equipment Name	Price	Installation Cost	Total	Total Price	Total Installation Cost	Total Total	Remarks for Order
100159	11/15/84	David Kent	OIS-G	PC9800	3500	150	3650	4200	250	4450	should be installed by 12/29/84
				B4670	700	100	800				
100160	11/20/84	John Wall	DB-G	Printer	1200	50	1250	6500	150	6650	to be received 12/15/84
				FDD	500	20	520				
				WP	4800	80	4880				

Fig. 4 Logical Form Schema Generation

recognizes its data structure and generates a corresponding logical form schema. The generated logical form schema is a data structure description with a form name at the top level. Each frame is mapped to a group item at the second level. An item is recognized when an item name is put into a field, and the item length is decided from the size of a (rectangular) field under (or to the right of) the name. The repeating number for a repeating (group) item is determined from the number of lines (or columns) under (or to the right of) the item name within the frame.

A frame itself may be a repeating group item, as Frame 2 in Fig. 2, or it can include another repeating group item, as Frame 3 in Fig. 4. As in Fig. 4, when a key word, such as Total, Average, Maximum or Minimum, is put into the table side (or table head, if horizontally repeating), new non-repeating items are generated (such as TotalPrice, TotalInstallationCost, TotalTotal), and a new repeating group item (such as Group 3G) is created. That is, some derived items (such as "vertical" totals) are automatically defined.

A logical form schema defines a logical form type, such as in Fig. 4. A logical form type can be considered as a non-first-normal-form relation. More than two levels of nesting are not allowed, however.

Users can give other logical form descriptions. These are derived items and their generation rules, data types, and relationship definitions. Derived item generation rules are defined, such as A=B+C, D=SUM(E) * 10, where A, B, C, D, E are item names. Functions, such as SUM, AVR, MAX, MIN, DATE, TIME and USER, can be used. The generation order for derived items is automatically recognized by the system, so users can define them in any sequence.

A relationship can be defined between two form types through related items. An index is created automatically on each item. Using relationships, a user can retrieve related forms without explicit join-like commands.

3.3 Physical Form Definition

A physical form schema is generated automatically from its logical form schema. A physical form instance is a set of variable length records. Each record consists of a number of variable length items, as shown in Fig. 6. Each item can be repeated any number of times. A physical form instance is a "flat" set of items, whereas a logical form instance is a structured set. Actually, as there is a mapping between logical and physical form items, a physical item ID does not directly correspond to a logical item ID (see Fig. 6).

Additionally, users may define indexes on any items. Indexes can be created at any time, even after the form instances have already been stored.

3.4 Structure Change

One of the outstanding PALET features is its flexibility. PALET allows external and logical form structure change at any time, even after the form instances have already been stored, without database reconstruction. (The logical structure change allowed include adding/deleting items, expanding/reducing item length, reordering of items, increasing/decreasing repeating number, etc.) The structure change is performed through the external form schema modification on the screen, as shown in Fig. 5. Users add/delete frames, or add/delete/expand/reduce/exchange rows or columns in the same manner as the external form schema definition. The corresponding logical form schema is automatical-

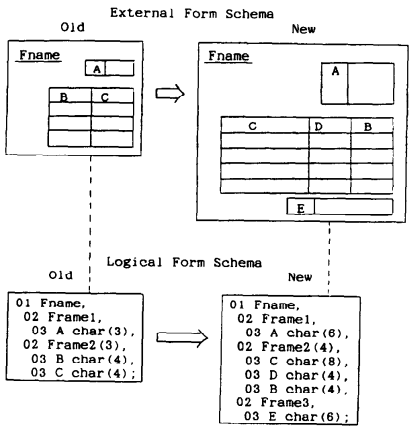

Fig. 5 Logical Form Schema Transformation

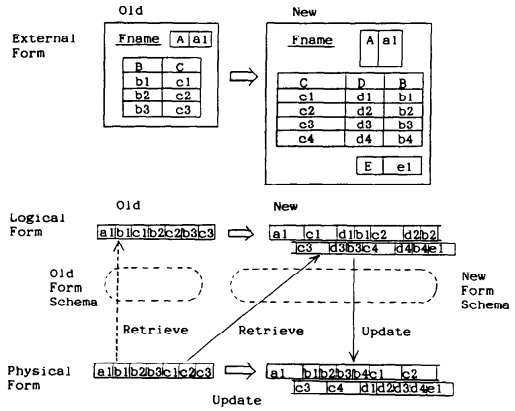

Fig. 6 Form Instance Transformation

ly changed when a user changes the external form schema (see Fig. 5). Therefore, it is necessary for the user to consider only the external forms; he does not have to handle logical (or physical) forms.

For the illustration in the upper left in Fig. 5, a user expands the row for item A, adds a new column for item D, expands the column for item C, exchanges columns for items B and C, adds a row for the second frame, and adds a new frame for item E, resulting in the illustration in the upper right of Fig. 5. These external level operations are interpreted to corresponding logical level operations; item length for item A is expanded, a new item D is added, item length for item C is expanded, data structures for items B and C are changed, the repeating number for group item Frame 2 is increased, and a new E is added.

External form schema operations and corresponding logical form schema operations (implicitly executed) are listed below. (Here, for simplicity, only vertically repeating items are assumed.)

[External Operations]		[Logical Operations]
Expand row/column	→	expand item length
Reduce row/column	→	reduce item length
Add/Delete row	→	increase/decrease repeating number
Add/Delete column	→	add/delete item, data structure change
Add/Delete frame	→	add/delete item
Exchange columns	→	data structure change
Add/Delete keyword to table side	→	increase/decrease repeating number, add/delete item, data structure change

The structure changes described above need database reconstruction for normal database systems, since database systems cannot change schemas dynamically during program execution. In the case of PALET, however, the logical data structure can be changed at any time with its screen format change. This can be performed through flexible physical-logical-external form mapping, and with variable length physical forms. Fig. 6 explains this mechanism (the same sample as in Fig. 5 is used).

In Fig. 6, an old form type is changed to a new one, when a user changes its external form schema. When the external form schema is changed, the structures for logical form instances are also changed, as a result of the logical form schema change. This is shown in the middle charts in Fig. 6.

Here, physical form instances remain unchanged, if no update occurs. When a user retrieves an old physical form through a new form schema, the old form is transformed into a new logical form (shown by the solid line from lower left to middle right). By this mapping, users can see old forms through a new form schema without any reconstruction (new item values are set to null, item lengths are adjusted). When the user, after the retrieval, inputs new item values and updates the form instance, the old physical form is changed into a new one created from the new logical form (shown by the solid line from middle right to lower right). Other physical forms are left unchanged. That is, old and new physical forms may exist in a document base at the same time, with no need for reconstruction.

4. Data Processing

4.1 Data Handling

PALET has several functions for use in manipulating form instances. These are form display, data entry, table processing, query, etc. To display a form instance, a mapping (physical-logical-external) is performed (see Fig. 2).

Normal table processing functions can be performed for each frame. These include: sort within a frame for rows/columns, move/copy/exchange for an item, move/copy/exchange/insert/delete for a row/column, copy for a form instance, etc. Null values are given

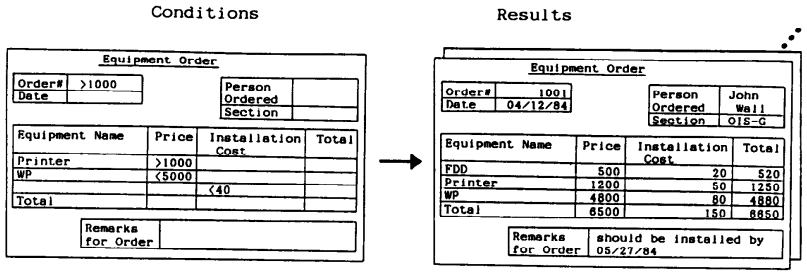

Fig. 7 Query for Forms

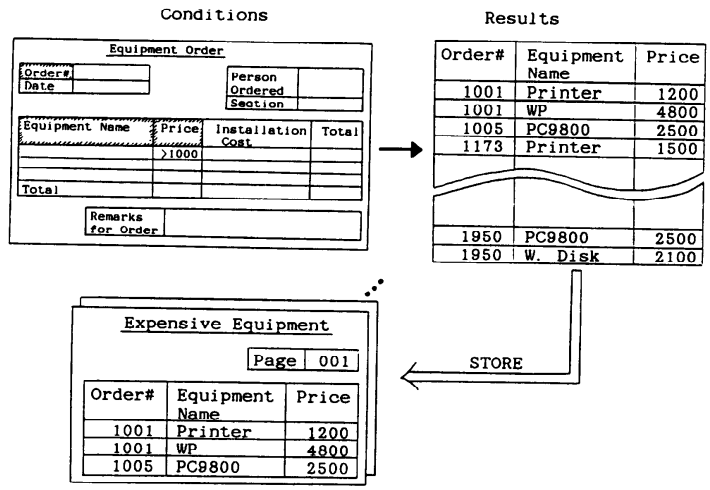

Fig. 8 Query for a Table

special consideration for their processing and calculation. For example, null rows are pushed downward while sorting.

PALET query functions are fairly simple, compared with those for relational database systems. However, PALET can retrieve forms in non-first-normal-form[4, 6]. Users can easily specify conditions for repeating group items on the screen. Fig. 7 shows an example of the query for forms. In the example, the conditions state:

"Find EquipmentOrder instances having
Order# > 1000 and
(EquipmentName, Price) ⊇
(Printer, > 1000) and
(EquipmentName, Price) ⊇ (WP, < 5000) and
InstallationCost ⊇ (< 40)."

As in the example above, conditions put in the same row constitute a set. And these sets (each searched independently) determine the semantics of a query. In Fig. 7, if "< 40" was put in the same row as "WP", the meaning would be (EquipmentName, Price, InstallationCost) ⊇ (WP, < 5000, < 40), and the first form instance in Fig. 7 would not satisfy this condition.

PALET has another type of query, called query for a table. This query makes it possible to retrieve form data in an unnested[6] format (first normal form). Fig. 8 shows an example. A user specifies conditions in fields as in the previous type of query. However in this type of query, multiple rows cannot be used in a frame. Additionally, the user indicates the items to be displayed in the result (projection; shown by hatching in Fig. 8). The system virtually flattens the form type (unnest[6]) to create a first normal form relation, searches each row to check the conditions, then extracts satisfying rows and projects them on specified items. The result is a flat table (optionally rows can be added for Total, Average, Maximum or Minimum; sorting is possible for the resulting table). Moreover, the resulting table can be stored as a form type (see the lower chart in Fig. 8). In this case, some number of rows (specified by the user) constitute a form instance (nest[6]). The system creates an item called a "Page" as a primary key, and generates its value for each instance.

4.2 Derived Form

Office workers often need summary reports in order to grasp the meaning of office activities and control them. A user can define a derived form type over some

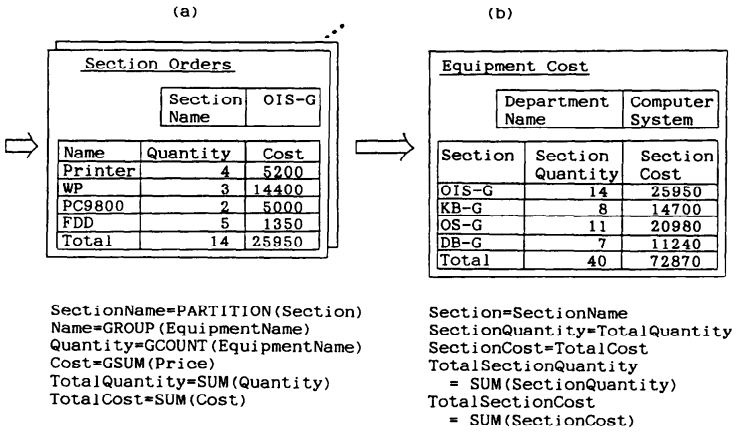

Fig. 9 Derived Form

other form type. The former includes the summary information for the latter. For example, assume that a department has several sections, and that each section issues a number of EquipmentOrders (Fig. 1). In this case, a user can define a derived form type, SectionOrders, over EquipmentOrder (see Fig. 9(a)). A SectionOrders instance contains all equipment names ordered by the section, where the same type of equipment is grouped and its quantity and cost is calculated.

A user defines a derived form type by specifying generation rules. The syntax given below is the superset of that for the derived item generation rules specified in 3.2.

J_k=PARTITION(I_t) or J_k=null

$J_s=F(I_1, I_2, \ldots, I_m)$ or $J_s=F(J_1, J_2, \ldots, J_n)$

($I_t(t=1, 2, \ldots, m)$ is an item for the source form, $J_s(s=1, 2, \ldots, n)$ is an item for the derived form, J_k is the primary key for the derived form, and F is a generation rule.)

Here, if PARTITION is specified for J_k, source form instances having the same value for I_t constitute a set, and for each set, one derived form instance is created (Fig. 9(a)). Otherwise, when J_k is a non-derived item, all source form instances are assembled to create a single derived form instance (Fig. 9(b) generated from (a)). Generation rule F includes simple copying of items, arithmetic operations, SUM, AVR, MAX, MIN, GROUP, GSUM, GAVR, GMAX, GMIN and GCOUNT. Group operations (from GSUM to GCOUNT above) must be accompanied by a single GROUP within a frame.

As a PALET special feature, a derived form type, freely designed by some user, may be stored as a set of physical forms. Therefore, a derived form type can be seen as a set of snapshots. Users can easily define higher level derived forms over the stored forms. A user may generate whole derived instances of a type at once, or he can generate a specific instance in order to modify an old one. Since a derived form type is physically stored, a user may add non-derived items for the derived form, and can modify these items as well as the snapshots of derived items.

Though the derived form is mainly for summary report snapshot generation, it can be used like a view (a view set is more appropriate, because the definition generates a set of form instances). In the simplest case, items for a derived form are a subset of items for a source form (a kind of subschema).

As a special function, an external form instance can be transformed into a text document, as well as into a graph/image document.

4.3 Remote Processing

PALET can be seen as a distributed document base management system. It can handle forms and text documents distributed over several offices (sites). The basic functions are remote query and electronic mail. Important features are: effective distributed information management using replicated directories and replicated form schemas, and remote form queries through freely designed screen formats.

A user can make a query for forms or for text documents existing in a remote document base. The retrieved forms/documents can be stored in the user's document base. Also, a user can send forms or text documents to other users residing in remote offices. Forms/documents received in a mailbox may be stored in a document base of the recipient.

In order to control the distributed document bases, PALET maintains a distributed directory in each office. Each directory has the same content (replicated). It contains mainly location and relationship information for offices, users, document bases, form types, and document groups. Form instances in a specific form type may be physically distributed to several document bases, as described in the directory. The system can find the location of any form, merely by checking the local directory. There is no need to consult other offices. This

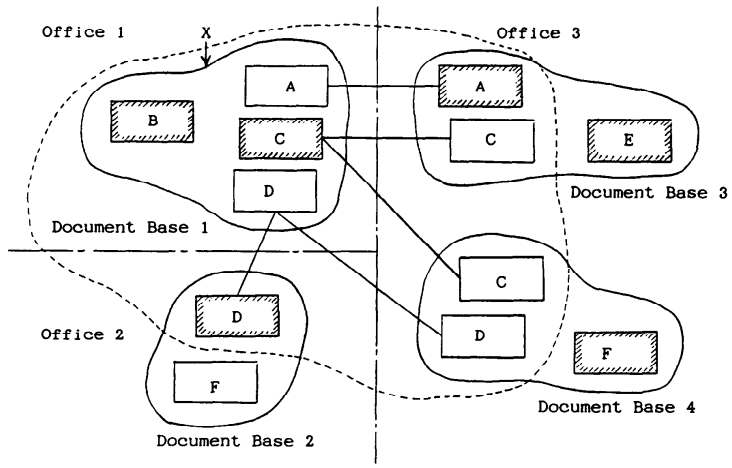

Fig. 10 A Scope

reduces the communication cost and shortens the response time for a remote query. For update activities, only form types are registered in the directory, not each form instance. Therefore, update frequency is not so high. A user registers shared form types to the directory. Other local form types cannot be seen by remote users.

A user can distribute a form schema in a form type to several document bases. The distribution enables scope management. The user scope consists of a set of form types to which he has access. The accessible form types are those local forms existing in the user's document bases, and those remote forms whose form schemas exist in his document bases. Fig. 10 shows the scope (within the dotted line) of user X in office 1, where rectangles show form types.

The form schema distribution provides flexible security control. For example, in Fig. 10, if the owner of form type E transfers its form schema to document base 1, the users of document base 1 will have access to E in document base 3. That is, a user distributes form schemas to other user groups (corresponding to document bases), who can use the form types. Using the form distribution, users can manage forms more securely than is possible when merely dividing them into shared and local forms.

The replicated form schema is utilized for various purposes. The form schema is needed for local processing, such as to handle received forms locally. Moreover, when making a query for remote forms, a local form schema is used to display and control them. This local processing reduces the communication cost and allows the remote query to be executed rapidly.

When replicated form schemas are distributed among offices, update control is necessary to avoid inconsistency. PALET identifies the copy/original and revisions of form schemas. The distribution source form schema is the original (shown by hatching in Fig. 10), and distributed form schemas are copies. Only the original form schema can be updated and re-transferred. Additionally, when an attempt is made to make a remote query or to transmit a form into the mailbox, revisions in the remote and local form schemas are checked so that no inconsistencies will occur. The copy/original ID and revision are included in the distributed directory to be checked locally, so that no excessive communication occurs.

5. Conclusion

PALET presents the following outstanding features, which enable office workers to speedily and flexibly design and handle miscellaneous office forms.

(1) Forms with relatively complex formats can be designed flexibly on the screen. Moreover, their basic logical and physical data descriptions are automatically generated from the screen formats interactively. Thus, the logical form in non-first-normal-form can be defined easily.

(2) The logical form structure (in non-first-normal-form) can be dynamically changed during program execution, through screen format change. This is possible even if the form data are stored, without any need for database reconstruction.

(3) Simple queries for forms in non-first-normal-form can be easily expressed on the screen. The query conditions may include repeating group items. Additionally, query for unnested forms (query for a table) is possible

(4) A derived form can be defined, which enables easy construction of summary reports. Higher level derived forms can be defined hierarchically.

(5) Remote processing, such as remote query and electronic mail, can be effectively executed with distributed directories, distributed form schemas, and scope management.

PALET can greatly reduce programming cost and data maintenance cost. As an example, a user may design an application form and its summary list to plan a sightseeing tour. He designs the forms using PALET, distributes the forms to remote offices, receives them with the contents filled in, and gathers them together into a summary list by the use of derived form functions.

PALET, with the functions described in this paper, is currently running on NEC small business computers. Local PALET functions are commercially available as NEC products (Japanese version only). Network functions are experimental.

Acknowledgment

The authors would like to thank the following NEC members: Asao Kaneko, who joined in the functional design of the PALET early version, and was particularly responsible for its user interface; Mitsuhiro Hattori, for his valuable suggestions for PALET basic architecture; Shin-etsu Kozuka and Hideki Hamura, who designed and refined the PALET extended version; Kazuhiko Honda, for his valuable discussions.

References

1. Czejdo, B. and Embley, D. W. Office Form Definition and Processing Using a Relational Data Model, *Proc. ACM SIGOA Conf.* (1984), 123-131.
2. Lefkovits, H. C. et al. A Status Report on the Activities of the CODASYL End User Facilities Committee (EUFC), *SIGMOD Record*, **10**, Nos. 2&3 (1979), 1-26.
3. Lum. V. Y. et al. OPAS: An Office Procedure Automation System, *IBM Systems Journal*, **21**, 3 (1982), 327-350.
4. Luo, D. and Yao, S. B. Form Operation by Example--A Language for Office Information Processing, *Proc. SIGMOD Conf.* (1981), 212-223.
5. Purvy, R. et al. The Design of Star's Records Processing: Data Processing for the Noncomputer Professional, *ACM Trans. Office Inf. Sys.* **1** (Jan. 1983), 3-24.
6. Schek, H. -J. and Pistor, P. Data Structures for an Integrated Data Base Management and Information Retrieval System, *Proc. VLDB* (1982), 197-207.
7. Shu, N. C. et al. Specifications of Forms Processing and Business Procedures for Office Automation, *IEEE Trans. Soft. Eng.*, **SE-8**, 5, Sept (1982), 499-512.
8. Tsichritzis, D. Form Management, *CACM*, **25**, 7 (July 1982), 453-478.
9. Yao, S. B. et al. FORMANAGER: An Office Forms Management System, *ACM Trans. Office Inf.* Sys., **2**, 3 (July 1984), 235-262.
10. Zloof, M. M. Office-by-Example: A Business Language That Unifies Data and Word Processing and Electronic Mail, *IBM Systems Journal*, **21**, 3 (1982), 272-304.

(Received May 21, 1985; revised November 22, 1985)