CONJUNCTIVE USE OF CONCEPTUAL FORM MODEL AND ACTUAL FORM GENERATOR

Masatake Nakanishi
Faculty of Business Management
Nagoya Keizai University
61-1 Uchikubo, Inuyama, Aichi, 484-8504 Japan

ABSTRACT

In view of the characteristics of contemporary data modeling and the limitations of the conventional form design method, Nakanishi proposed a Conceptual Form model as a new model for the basis of the form design theory in 1998. This model is a conceptual abstraction of external schema. Conceptual Form Formula, which comprises the model, is an algebraic representation of the Conceptual Form and is determined by the logical data structure of the target data source and its selected entity access path. This formula enables us to grasp the whole form patterns derivable from given data source. In contrast, many of actual form generators seem to lack of this kind of pattern analysis.

This paper proposes an idea of conjunctive use of the Conceptual Form model and actual form generator for obtaining a good productivity with reliability, and explains the experimental result of a concrete application case.

Keywords: Conceptual forms, Database, User interfaces, Information Systems Design

CONCEPTUAL FORM MODEL

In the database modelling field, extensive research works have been devoted to the study of visual query systems (VQSs). Most of their motivations are placed to aim at providing both a language to express the queries in a visual format and a variety of functionalities to facilitate user-system interaction. Many of the VQS proposals are based on diagrammatic representations of semantic data models, i.e. Entity-Relationship model. As Shoshani (1978) claimed in his proposal of CABLE, “if relationships are known, the use should not be required to specify them in the data language.” Elmasri and Larson (1985) identified a requirement of *User tailorability*, that is, “the interface should allow the user to view the database in the way the user is most comfortable with when formulating a particular query, and to proceed through query formulation differently.” These kinds of sense may have been common in many VQS researchers.

As to the logical form structure, Tsuruoka et al. (1985) presented an idea of office form schema structure which consists of an external form, logical form and physical form. Kitagawa et al. (1989) precisely studied a language of office form processing based on the nested table presentation. Santucci et al. (1997) proposed a form-based visualization for arranging the query result in a nonflat table. Gyssens et al. (1996) investigated the variety of flat table expression of query output.

Many research considerations begin with access path entry which user has already chosen. If it is possible to supply theoretical list of form patterns to the user, all he has to do becomes simple conduct of form choice and its customization. However, there have been few systematic researchers which are devoted to clarify certain relationship of possible form pattern and topological structure of database accessing field.

Nakanishi (1998) introduced the idea of *Conceptual Form* model as a new model for the basis of the form design theory. *Conceptual Form* is a theoretical construct of external schema, which shares data structures with a conceptual data model. Possible Patterns to generate *Conceptual Forms* are determined by the logical data structure of the target data source and its selected entity access paths. We are able to create supporting tools for requirement analysis and screen/report form design, utilizing the catalogued patterns to generate *Conceptual Forms*.

Conceptual Form model is a conceptual abstraction of external schema, which is a unit of logical data presentation freed from implementation technique constraints such as graphical user interfaces, screen/report page sizes, presenting dimensions, positioning, or visual effects. For example, when designing physical screen/reports that use master (*A*) to detail (*B*) entity access path, we have to apply implementation techniques to pack the required data presentation onto the given physical area. However, in *Conceptual Form* designing, we will be free from such physical constraints and able to obtain a simple form pattern denoted as $A\vec{B}$ (see **Figure 1**) .

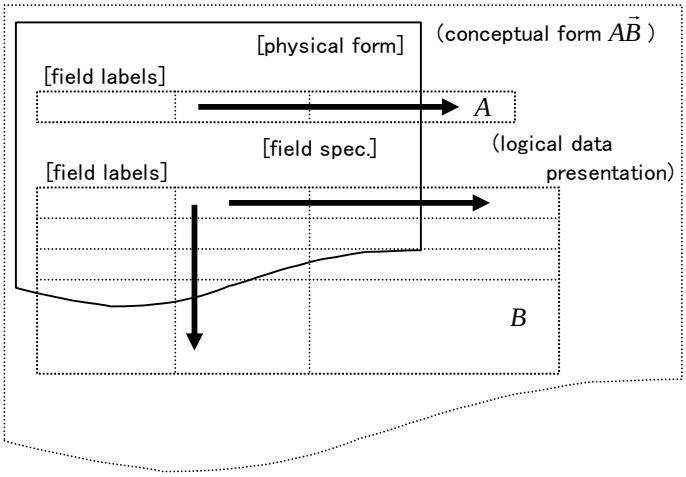

Figure 1. Physical Form and 'Conceptual Form'

Since screen/report forms are media for handling data in connection with the database, their form structures should have some projected images of the target data source. Now, we will suppose that the target data source of *Conceptual Form* is the conceptual data model consisting of adequately normalized entity-relationships.

There are three factors to determine the structure of this *Conceptual Form*.

(i) Topological structure of the target data source
Possible access path patterns are determined by the topological structure type.

(ii) Access path patterns
Every selected access path pattern has its unique standard data presentation structure, which is formularized by *Conceptual Form Formula*.

(iii) Derivative data presentation structure
Derivative data presentation structures on the selected access path patterns are logically derived from the standard form with the formula.

Other terms are defined as follows:

- *Access Path Reading*

Entity data reading by using a selected entity access path to generate a *Conceptual Form*.

- *Data Source Type*

Topological type of the fragment data model to be accessed by the *Conceptual Form*.

- *Conceptual Form Formula*

Algebraic representation of *Access Path Reading* to generate its own *Conceptual Form*.

- *Conceptual Form Generation Pattern*

One of the data structure patterns of the *Conceptual Forms* determined by the selected *Data Source Type*.

- *Row Block*

A displayed unit of the entity type on the form, which has a property of single/multi-rows.

Data Source Types

In actual screen/report form design, the number of entity types accessed by each form is under 6 in majority cases viewed from a practical standpoint. Then a study was conducted to list up all the topological types of the data source which has less than 6 entity types.

As a result, 50 basic *Data Source Types* were obtained. **Figure 2** shows a mandala-like map of the *Data Source Type*s, which visually presents the topological similarities and symmetries among those types. There are 19 type pairs (38 types), whose members are

topologically symmetrical with each other, for example: V_3 and ${}^tV_3$. This set of basic *Data Source Types* gives a universe of discourse for considering the closure of *Conceptual Form* generation. Nakanishi (2002, 2003a) made up the catalogue of all the *Conceptual Form Generation Patterns* for every basic *Data Source Type*. Thus, we obtain the theoretical basis for establishing a practically automated form generation tool.

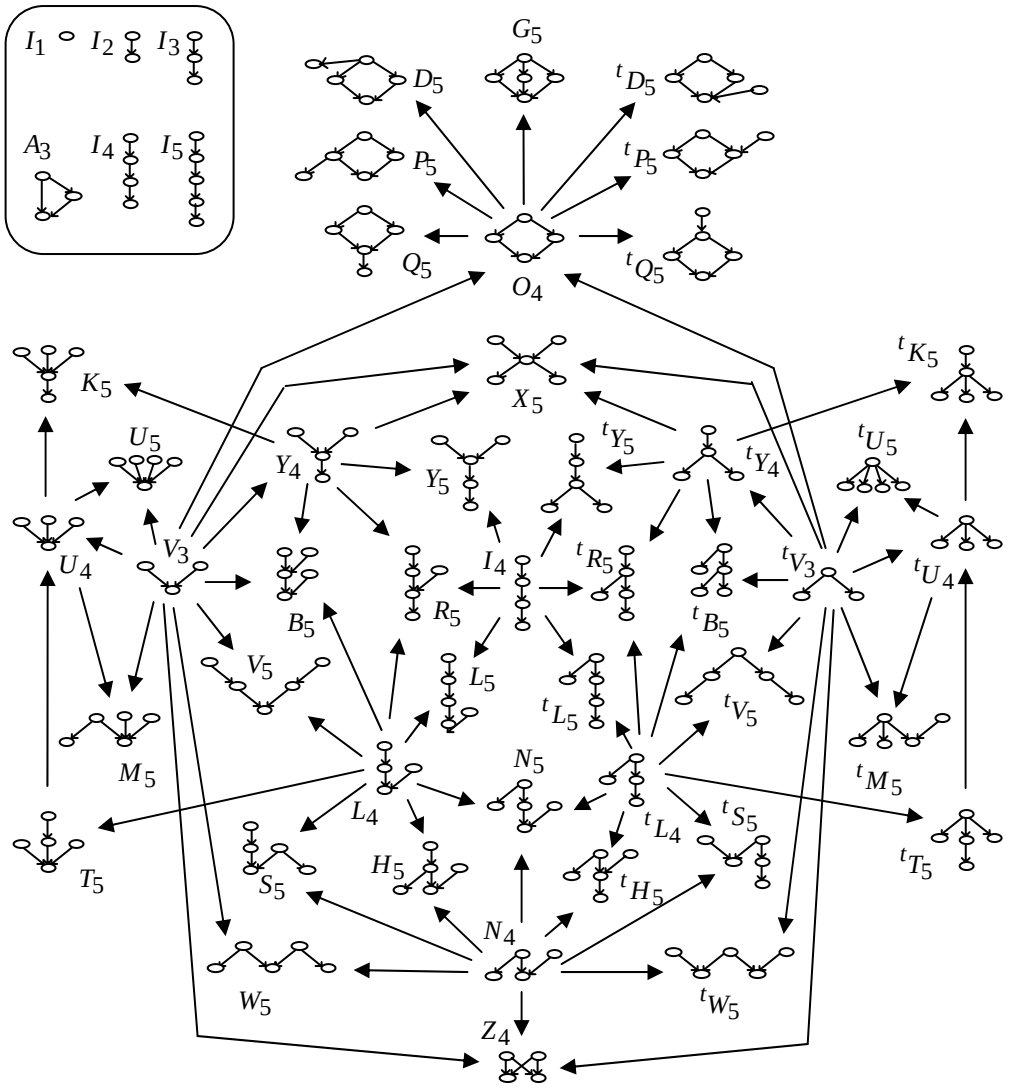

Figure 2. Topological Map of 50 Basic Data Source Types

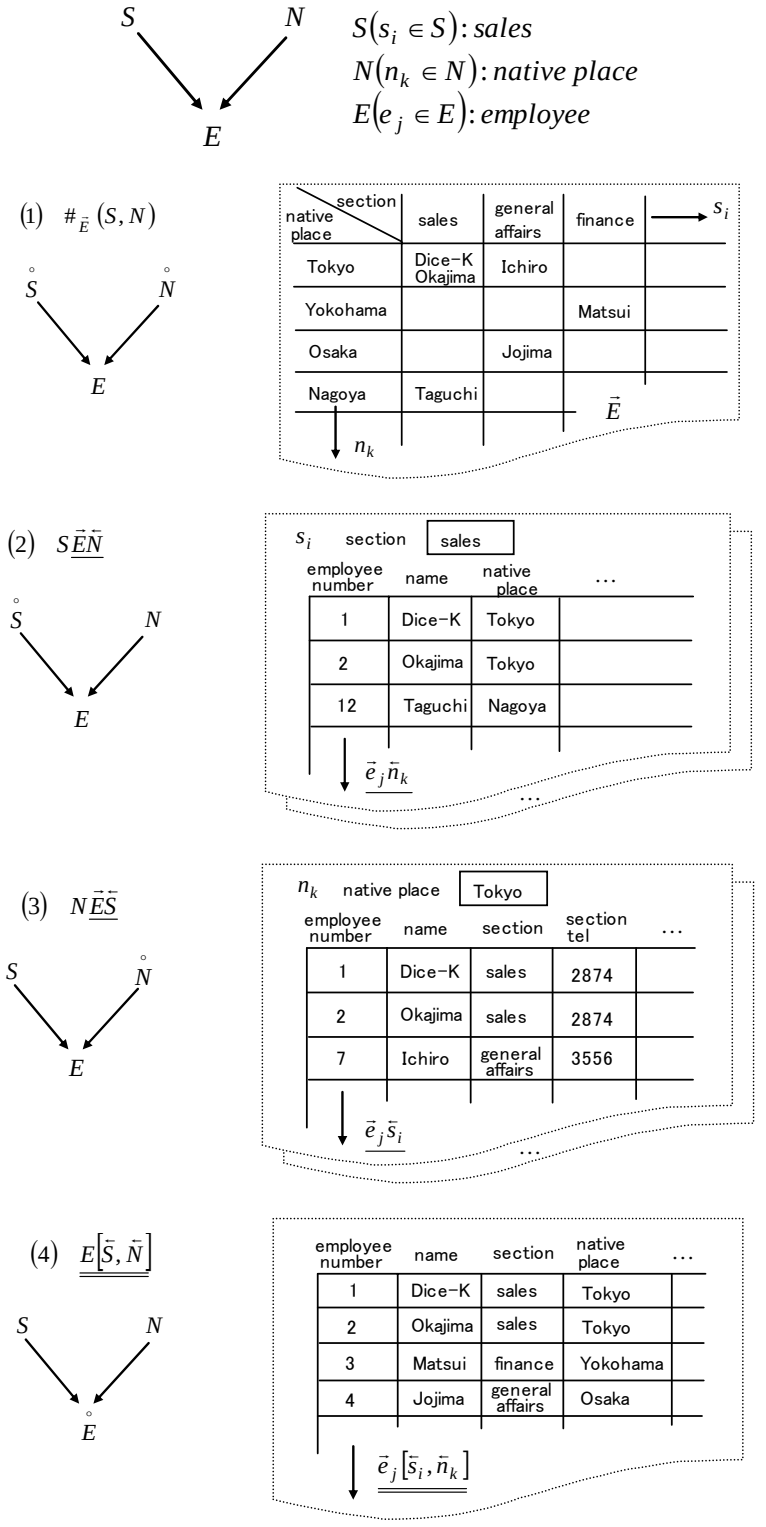

Figure 3. Example of V_3 type Data Source and its Conceptual Form Patterns

Conceptual Form Generation Patterns

Figure 3 shows a simple ER diagram of a V_3 type data source and its *Conceptual Form Generation Patterns*. This *Data Source Type* holds V figure and 3 elements. Here, each entity/relationship type is denoted by an alphabetical symbol, and one-to-many cardinality of referencing path by a single head arrow. This data source possesses 4 access patterns, whose logical presentation image are illustrated as follows, where ◦ is written above the symbol of access entry entity in each simple ER diagram.

As we have already discussed the calculation rules of the form formula in detail (Nakanishi, 1998), we would like to illustrate them by simple examples of basic form patterns in this paper. **Figure 4** shows a simple data map of ER diagram, which represents a conceptual data model. A fragment data model to be accessed by certain *Conceptual Forms* is extracted from this model as the target data source. If all the entity/relationship nodes are extracted, then the target data source belongs to W_5 type; if not, then it belongs to a subset *Data Source Type* of W_5 type such as V_3 type. Database user can select a preferable access path pattern from those determined by the *Data Source Type*. Then he can immediately obtain the *Conceptual Form Formula* of the standard form structure in line with his selection.

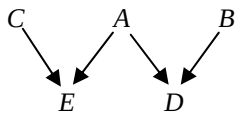

Figure 4. Example of Simple Data Map

Downward/Upward Path Reading

We have two kinds of opposite direction in path reading: *Downward* or *Upward*.

a) Downward Path Reading

An entity data reading on the referencing path between *A* and *D* shall be from the primary key of an entity occurrence $a_i \in A$ to the foreign key of $d_j \in D$. We will call it the *Downward Path Reading*, where *A* performs the preceding node and the following node *D* is written as $\vec{D}$ in the formula. In $A\vec{D}$, whose target *Data Source Type* is I_2, row blocks of *A* and $\vec{D}$ may hold a master-detail association (**Figure 5**).

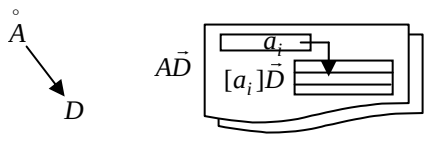

Figure 5. Example of Downward Path Reading

b) Upward Path Reading

An entity data reading on the same referencing path shall be from the foreign key of $d_i \in D$ to the primary key of $a_j \in A$. We will call it the *Upward Path Reading*, where D performs the preceding node and the following node A is written as $\bar{A}$ in the formula. In $D\bar{A}$, non-key attributes of $\bar{A}$ are partially or transitively dependent on the primary key of A. As each single row a_j is determined by each d_i, let the row pairs be denormalizingly joined into a new row on the form. We will call it the *1st level Row Joining*, where the created multi-row block is written as $\underline{D\bar{A}}$ in the formula (**Figure 6**).

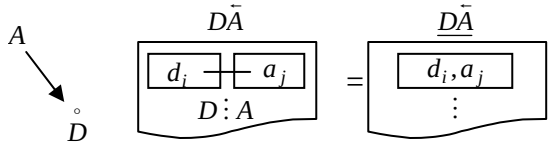

Figure 6. Example of Upward Path Reading

Parallel Strings

Let the several entity associations be connected as a junction of *Divergence* or *Convergence*. We will call it the *Parallel Strings*. The cases **c)**, **e)** are examples of *List Group Presentation*, and **d)**, **f)** of *Cross Reference Presentation*, each of which is derived through reading of the *Parallel Strings* respectively.

c) Parallel Downward Divergence

Let $A\vec{D}$ and $A\vec{E}$ be divergently connected at the entry point A as a junction. We will call it the *Parallel Downward Divergence* and formalize it as $A\left[\vec{D},\vec{E}\right]$, where master block A possesses two detail blocks of $\vec{D},\vec{E}$. The topological type of its target data source is ${}^{t}V_3$, which has transposed figure of V_3 (**Figure 7**).

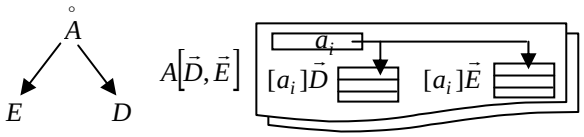

Figure 7. Example of Parallel Downward Divergence

d) Parallel Upward Convergence

Let $D\bar{A}$ and $E\bar{A}$ on the ${}^{t}V_3$ type data source be convergently connected at the intersection point $\bar{A}$ of the coordinate axes D and E. We will call it the *Parallel Upward Convergence*. Each intersection cell shows the $\bar{A}$ occurrence, single-row block,

associated with D and E which may be realized as a pop-up image, because the foreign keys of $a_k \in A$ is imposed to maintain referential integrity with primary keys $d_i \in D$ and $e_j \in E$ respectively (**Figure 8**).

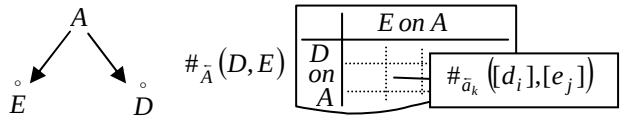

Figure 8. Example of Parallel Upward Convergence

e) Parallel Upward Divergence

Let $D\vec{A}$ and $D\vec{B}$ be divergently connected at the entry point D as a junction. We will call it the *Parallel Upward Divergence* and formalize it as $D\left[\vec{A}, \vec{B}\right]$ in the formula. As single rows a_j, b_k are determined by each d_i respectively, let the row combinations be denormalizingly joined into a new row on the form. We will call it *the 2nd Level Row Joining*, where the created multi-row blocks is written as $\underline{\underline{D\left[\vec{A}, \vec{B}\right]}}$ in the formula. The process of *2nd Level Row Joining* is logically derived from the *1st Level* (**Figure 9**).

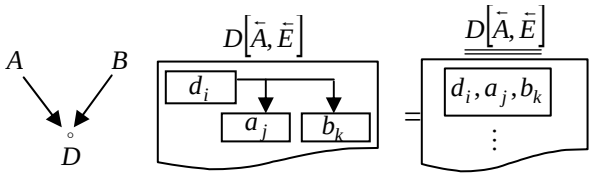

Figure 9. Example of Parallel Upward Divergence

f) Parallel Downward Convergence

Let $A\vec{D}$ and $B\vec{D}$ on the V_3 type data source be convergently connected at the intersection point $\vec{D}$ of the coordinate axes A and B. We will call it the *Parallel Downward Convergence*. Each intersection cell shows the $\vec{D}$ occurrences, multi-row block, associated with A and B which may be realized as a pop-up image (**Figure 10**).

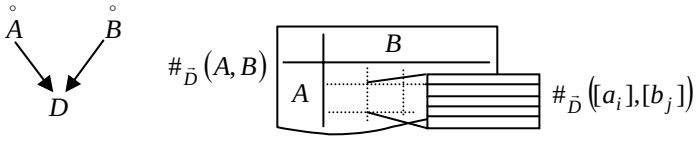

Figure 10. Example of Parallel Downward Convergence

EXAMINATION OF FORM GENERATION BY CONCEPTUAL FORM ENGINE

Problem of Invisible Constraints in Form Generation Tool

Many of actual form generators seem to lack pattern analysis of the whole form patterns derivable from given data source. For example, according to Nakanishi's research (2003b), form generator embedded in *MS Access* is able to function its automated generation for only one-third of the whole basic *Data Source Types*. As the other two-third types contain ${}^{t}V_3$ type in them as their subtype without exception, this tool seems to avoid some complicated denormalization with multiple detail parts caused by ${}^{t}V_3$ structure. Unfortunately since users are not given any reason of rejection of such form generation by the tool, it is likely that they may loose chance to know the possibility of other valuable forms. We will call such an uninformed rejection *Invisible Constraints in Form Generation Tool*.

Conceptual Form model enables us to grasp the whole form patterns derivable from given data source. If we utilize this model and give the form pattern information to the users, and transform the users' choice into parameters for driving a good form generator, then we are able to make users freed from those *Invisible Constraints*.

In this research, we examined the satisfactory power of such a theoretical middleware, which we call *Conceptual Form Engine* based on a conjunctive use of *Conceptual Form Model* and actual form generator, and thus which bridges practical database with practical form generator.

Experimental Condition of Form Generation Environment

Our experimental condition of form generation environment was as follows:

- Data source: *MS Access* tables (W_5 type and its subset types such as V_3 and ${}^{t}V_3$).
- Data source information: database definition data of *MS Access*.
- *Conceptual Form Engine* : created by the author with *MS Excel + VBA*.
- Form generator: *FormEditor + DocCreator* in *CoReports ver.9* by HOS system inc..

In the beginning, *Conceptual Form Engine* sends SQL to *MS Access* for gathering data value and database definition data. The engine analyzes the received data to clarify the whole derivable form patterns from the given data source and informs the result to the user. Through observing and considering the derivable form patterns, he may give ad-hoc requirement on form generation, then the engine automatically provides the necessary parameters for driving *CoReports* to generate an initial specification of actual form to be used in prototyping process by the user. *FormEditor* is a schema-free form designer aid – independent from database schema structure -- which enables users to create form files

with precise form image. *DocCreator* works to map each data item to each data field of the target form file respectively.

We had three reasons to select *CoReports* as our test tool:

- It gains a good reputation among practitioners of information systems development
- It is easy to create the middleware interface upon it because of its schema-freeness.
- Its XML meta-model enables us to evaluate its expressive power objectively.

Under the status-quo, *CoReports* does not contribute to users about the knowledge of derivable form patterns, and so they are forced to conduct some trial and error approach in form designing. However, once it is connected with *Conceptual Form Engine*, users no longer burden such a helpless work load.

Examination of Seamless Procedure of Form Generation

We established the following seamless procedure of form generation to be examined the validity, reliability and productivity.

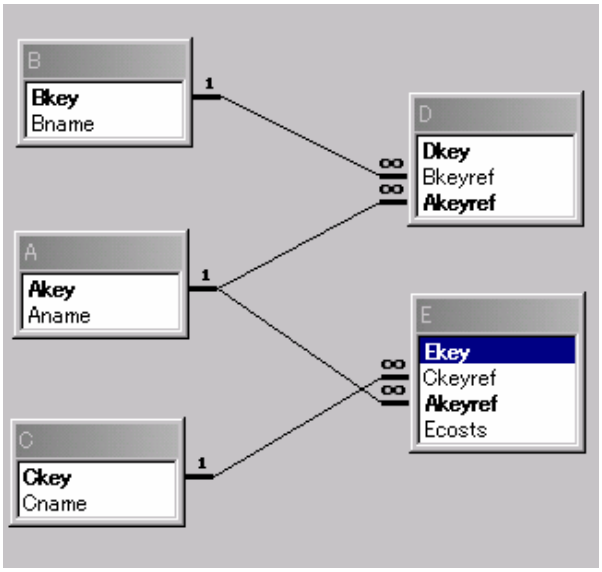

Figure 11. Test Data Source for Examination

We provided a test data source consisting of five tables *A*, *B*, *C*, *D* and *E*, whose primary keys are *Akey*, *Bkey*, *Ckey*, *Dkey* and *Ekey* respectively. *Akey*, *Bkey* and *Ckey* are connected with foreign keys *Akeyref*, *Bkeyref* and *Ckeyref* of *D* and *E* via one-to-many referencing paths respectively (**Figure 11**). This *Data Source Type* W_5 contains various types of subset $N_4, V_3, {}^tV_3, I_2$ and I_1. We remarked how W_5, N_4 and ${}^tV_3$ were resultantly

processed because these data source types contain ${}^{t}V_3$ as subset in them, which was uninformedly rejected by *MS Access* form generation tool.

User was allowed to select any subset of tables as his accessing data source. The *Conceptual Form Engine* inspects the validity of user selection, and if some invalid selection is found, then the engine automatically notifies it to him, searches for readable fragment path, and shows the result to the user.

Entry	C	Table name	item names-->	selected tables	4	available tables	2	
Table1	1	A	Akey	Aname				
Table2	1	B	Bkey	Bname				
Table3	E	C	Ckey	Cname				
Table4		D	Dkey	Bkeyref	Akeyref			
Table5	1	E	Ekey	Ckeyref	Akeyref	Ecosts		
						Conceptual Form Formula		
		B is disjointed !				C >E <A		

Figure 12. Validation Check and Resultant Generative Form

Figure 12 illustrates the validation check by the *Conceptual Form Engine* on selected four tables *A*, *B*, *C* and *E*. The engine notifies the disjointed table *B* from others because of lack of *D*, and returns the generative form pattern $C\vec{E}\vec{A}$ ('C>E<A' denoted in *MS Excel* because of its expressive power) in *Conceptual Form Formula*. In this case, two tables *E* and *A* are available except the entry table *C*.

Table3		View1	Ckey			
Ckey	Cname	Ekey	*Ckeyref*	Akey	Ecosts	Aname
1	2	3	7	5	4	6
c1	c1 name	e1	c1	a1	1	a1 name
c1	c1 name	e4	c1	a3	7	a3name
c2	c2name	e2	c2	a2	5	a2name
c3	c3name	e3	c3	a1	2	a1 name
c3	c3name	e5	c3	a2	12	a2name

Figure 13. Conceptual Form Structure with Data Values

Figure 13 illustrates the *Conceptual Form* Structure of $C\vec{E}\vec{A}$ with data values, where C and $\vec{E}\vec{A}$ become the master and detail part of the purposed form respectively; $\vec{E}\vec{A}$ is obtained from *E* and *A* as a view dependent on *C*. Here, as the preparation for making output data file, the user is allowed to change the order of data items given from each table into his preferable order (see 3rd row of **Figure 13**). When in completing this process, *Conceptual Form* Engine automatically creates the three data files: 1) output data file (CSV format), 2) *FormEditor* parameter file (XML format), and 3) *DocCreator* parameter file (XML format).

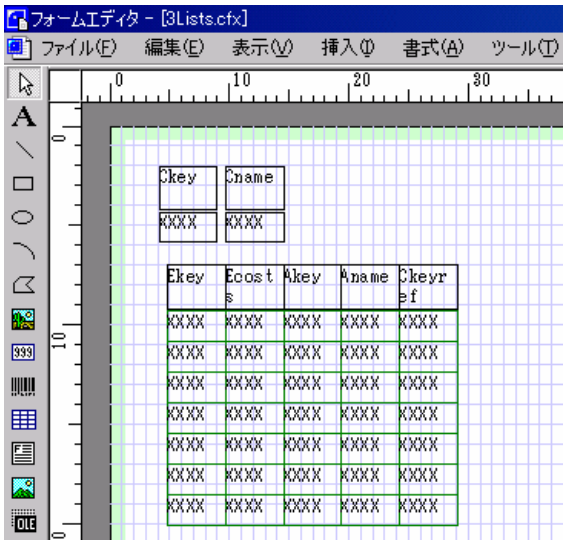

Figure 14. Initially Generated *FormEditor* File

Figure 14 illustrates an initially generated *FormEditor* file, where data fields of master parts are separately located according to the user's preference. User is allowed to modify layout, labelling, fonts, lines, colouring, and page controls of the form with drawing/editing function of *FormEditor*. However, every connection between each data field in form file and each data item in output data file is already defined by the *Conceptual Form Engine* and is identically being observed until finalized.

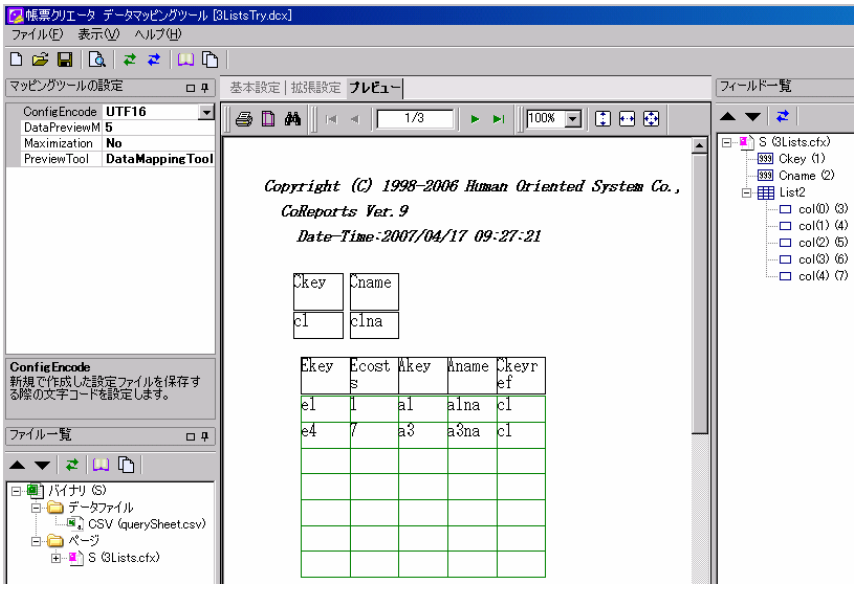

Figure 15. Automated Mapping of *FormEditor* File and CSV Data File

Figure 15 illustrates the automated mapping of *FormEditor* File and CSV data file. Since *Conceptual Form Engine* simultaneously creates the output data file, the initial form file and the mapping file as bondage set, mapping of form file and output data file is automatically guaranteed. Therefore, user has nothing to do about mapping in this session. If user is not satisfied with this result, he may restart the form generation process from any fall-back points.

RESULT OF EXAMINATION AND EVALUATION

Since *CoReports* does not possess the satisfactorily expressive power on cross reference table, entity path reading with multiple-entry is eliminated from our scope of examination. **Figure 16** depicts the whole form patterns of single-entry path reading on W_5 and its subset types $N_4, V_3, {}^tV_3, I_2$ and I_1. In this research, our form generation toll successfully generated the target actual forms for all the given *Conceptual Form Generation Patterns*. In contrast, *MS Access* form generator did not work its automated generation function for $W_5, N_4, {}^tV_3$, which contained ${}^tV_3$ as a subset type inside them.

W_5 type	(1) $E\left[\vec{C}, \vec{A}\vec{D}\vec{B}\right] = E\left[\vec{C}, \vec{A}\right]\vec{D}\vec{B}$ (2) $D\left[\vec{B}, \vec{A}\vec{E}\vec{C}\right] = D\left[\vec{B}, \vec{A}\right]\vec{E}\vec{C}$ (3) $A\left[\vec{D}\vec{B}, \vec{E}\vec{C}\right]$ (4) $B\vec{D}\vec{A}\vec{E}\vec{C}$ (5) $C\vec{E}\vec{A}\vec{D}\vec{B}$	V_3 type	(1) $E\left[\vec{C}, \vec{A}\right]$ (2) $D\left[\vec{B}, \vec{A}\right]$ (3) $A\vec{D}\vec{B}$ (4) $A\vec{E}\vec{C}$ (5) $B\vec{D}\vec{A}$ (6) $C\vec{E}\vec{A}$	${}^tV_3$ type	(1) $D\vec{A}\vec{E}$ (2) $E\vec{A}\vec{D}$ (3) $A\left[\vec{D}, \vec{E}\right]$
N_4 type	(1) $E\left[\vec{C}, \vec{A}\vec{D}\right] = E\left[\vec{C}, \vec{A}\right]\vec{D}$ (2) $D\left[\vec{B}, \vec{A}\vec{E}\right] = D\left[\vec{B}, \vec{A}\right]\vec{E}$ (3) $E\vec{A}\vec{D}\vec{B}$ (4) $D\vec{A}\vec{E}\vec{C}$ (5) $A\left[\vec{D}\vec{B}, \vec{E}\right]$ (6) $A\left[\vec{D}, \vec{E}\vec{C}\right]$ (7) $B\vec{D}\vec{A}\vec{E}$ (8) $C\vec{E}\vec{A}\vec{D}$	I_2 type	(1) $E\vec{C}$ (2) $E\vec{A}$ (3) $D\vec{B}$ (4) $D\vec{A}$ (5) $A\vec{D}$ (6) $A\vec{E}$ (7) $B\vec{D}$ (8) $C\vec{E}$	I_1 type	(1) A (2) B (3) C (4) D (5) E

Figure 16. Form Patterns of Single-Entry Path Reading on W_5 and Subset of W_5

The result of this examination verified the practicality of conjunctive use of *Conceptual Form Model* and Actual Form Generator. There still does not exist such a standard language which describes form generation patterns, different from the database area which has SQL as the standard. The *Conceptual Form Model* can take the role of the required description language. The model is featured as independent from physical factors, and can

contribute to bridging a gap between actual database and actual form generator through its formula transformation power. The *Conceptual Form Engine*, whose core is the *Conceptual Form Model*, extracts the logical data structure from the given data source, determines the derivable form patterns through formula transformation, and creates the parameters for the target form generator. On the basis of the *Conceptual Form Engine* which is used as a middleware, database and form generator are able to become much flexibly interoperable with one another.

CoReports will enhance the expressive power of cross reference table in coming autumn, when we project to examine the form generation for such forms.

REFERENCES

Elmasri, R., and Larson, J. (1985), A Graphical Query Facility for ER Databases, in *Entity-Relationship Approach – The Use of ER Concept in Knowledge Representation*, North-Holland: 236-245.

Gyssens, M., Lakshmanan, L.V.S., and Subramanian, I.N. (1996), Tables as a Paradigm for Querying and Restructuring, in *Proceedings of ACM Symposium on Principles of Database Systems (PODS '96), Montreal*.

Kitagawa, H., and Kunii, T. (1989), *The Unnormalized Relational Data Model – for Office Form Processor Design*, Springer-Verlag.

Nakanishi, M. (1998), An Algebraic Analysis of Form Structures with a 'Conceptual Form Model', in *Journal of Japan Society for Management Informations*, 6(4): 49-61.

Nakanishi, M. (2002), On a Cataloguing of Data Source Types, Conceptual Forms Generation Patterns, and Conceptual Forms Templates, in *Journal of Japan Society for Management Informations*, (11)1: 41-67.

Nakanishi, M.(2003a), A Proposal of 'Conceptual Form' and its Application to a Form-Generating Catalogue, in *Essays & Studies in Commemoration of the Establishment of the Faculty of Business Administration*, Nagoya Keizai University: 203-225.

Nakanishi, M.(2003b), Evaluating Form Generation Function of DB Tool by Conceptual Forms Model – a Case Using Microsoft Access, in *Proceedings of Symposium of Japan Society for Management Informations, spring.*.

Santucci, G., and Tarantino, L. (1994), A Dynamic Form-Based Data Visualiser for Semantic Query Languages, in *Proceedings of 2nd International Workshop on Interfaces to Database Systems*: 249-265.

Shoshani, A. (1978), A Language Based on the Entity-Relationship Model, in *Lawrence Berkeley Laboratory*, no.22033.

Tsuruoka, K., Watabe, K., and Nishihara, Y. (1985), PALET: A Flexible Office Form Management System, in *Journal of Information Processing*, (8)4: 280-287.