Decision Support Systems 36 (2004) 313–335
Decision Support Systems
www.elsevier.com/locate/dsw

A methodology for designing form-based decision support systems

Jen-Her Wu[a,*], Her-Sen Doong[a], Ching-Chang Lee[b], Tse-Chih Hsia[c], Ting-Peng Liang[a]

[a] *Department of Information Management, National Sun Yat-Sen University, Hsi-Tze Wan, Kaohsiung 80424, Taiwan*
[b] *Department of Information Management, Shu-Te Institute of Technology, Yen Chau, Kaoshiung County, Taiwan*
[c] *Department of Business Management, Mei-Ho Institute of Technology, Neipu, Ping Tung, Taiwan*

Received 1 May 2002; accepted 26 August 2002

Abstract

Form-based decision support systems (FBDSS) are special types of information systems that use forms to present information for decision-making. Paper forms that are often used for collecting and disseminating information in offices are natural bases for eliciting user information requirements. They are particularly useful for user-oriented systems, such as executive information systems and DSS. The paper presents a methodology that uses factoring and synthesis to process knowledge involved in forms for designing FBDSS. The resulting system allows flexible creation and modification of computer-generated forms useful for decision-making. A prototype system illustrating the method is described.

Keywords: Form-based decision support systems; Form management; System analysis and design; End-user computing

1. Introduction

A form is a way of organizing and presenting data. Most organizations use forms to present data and to communicate with their affiliated entities in the office environment. Previous research has indicated that forms are the central structural abstraction for data collection, storage, retrieval and updating in an office [9,10,13]. Data in forms are often well structured and can easily be formalized [3]. Therefore, forms are an important means to designing and developing user-oriented information systems, such as executive information systems (EIS) and form-based decision support systems (FBDSS) [6].

Previous research has studied several approaches to using business forms for analyzing office activities and determining information requirement. For instance, Tsichritzis [10] introduced the concept of form types, form templates and form instances when forms are used to integrate different facilities and services in office information systems. Shu et al. [8,9] proposed using forms to specify system requirements. Others investigated the use of forms to generate entity relationship diagrams, determine functional dependencies and present information [1–5,8,9,11,12]. Table 1 shows a summary of their work.

In addition to database design and system analysis, forms are also useful for information presentation in decision support systems [6]. To meet the dynamic needs of decision makers, DSS must have flexibility in information presentation, which includes flexibility in adding new data items, modifying data items,

* Corresponding author. Tel.: +886-7-5252000x4722; fax: +886-7-5254799.
E-mail address: jhwu@mis.nsysu.edu.tw (J.-H. Wu).

0167-9236/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0167-9236(02)00145-8

Table 1
Previous research in using forms for information system design

Research	Objective	Result or system output
Tsichritzis [10]	To integrate different facilities and services in Office Information Systems	(1) A facility for the specification and implementation of automatic form procedures (2) Techniques for the analysis of office flow
Shu et al. [9]	To decompose business functions into meaningfully connected form processes for office automation	(1) A form processing language (2) A business procedure definition language
Batini et al. [1]	Database analysis and design	Generate entity relationship diagrams from forms
Choobineh et al. [4,5]	Database analysis and design	Generate entity relationship diagrams from forms
Choobineh and Venkatraman [3]	Database analysis and design	Determine functional dependencies
Wu [11,12]	Flexible form presentation	Facilitate flexible form presentation

changing the layout of existing forms or creating an entirely new form. A form-based DSS organizes the result in tables or in more general lists that can easily be scrolled. Traditionally, forms are predefined at the system analysis and design stage when the software is developed. Rewriting system code is necessary when a new form is desired or an old one needs to be revised. This is inconvenient, expensive and significantly restricts the flexibility of a system.

One way to overcome the problem is to develop systems that allow users to create and modify their forms conveniently. In other words, end users can handle some of their own information reporting needs through ad hoc form construction without the intervention of computer professionals. This flexibility is particularly useful in developing executive information systems and user-oriented decision support systems when ad hoc analysis is essential [8,9,10,13].

In this paper, a methodology for designing FBDSS is presented. A prototype system capable of providing flexible two-dimensional form presentations of numerical data, called Flexform, is implemented for demonstration. Flexform was designed for the Department of Transportation of Taiwan Government (DTTG) to support the analysis of traffic accidents.

Although report generation is not new, most existing approaches require that the user have knowledge of the database. The role of the system is to organize the specified data into tables. For instance, Microsoft ACCESS provides the functions of *Auto Report*, *Report Wizard* and *Design View* that allow end users to manipulate their one- and two-dimensional forms, respectively. *Auto Report* and *Report Wizard* allow end user to specify the data source and follow the instructions to create a fixed one-dimensional form. *Design View* provides more sophisticated capabilities, such as allowing creation of subreports within a report and manipulation of more complex two-dimensional forms. However, end users need to be able to define data sources and understand the database schema before they can integrate multiple tables or aggregate the desired data. This is difficult for most users who may not have professional database knowledge on its schema. Our approach embeds more semantic knowledge in forms in the factoring and generalization process to greatly simplify the ad hoc reporting process.

The remainder of the paper is organized as follows. Sections 2 and 3 define various concepts associated with form management, including a form, template and meta-template. Section 4 describes the factoring and synthesis of forms. Section 5 presents the methodology for designing FBDSS. Section 6 shows the Flexform system and an example to illustrate the process of flexible form generation. The last section concludes the paper.

2. Characteristics of forms

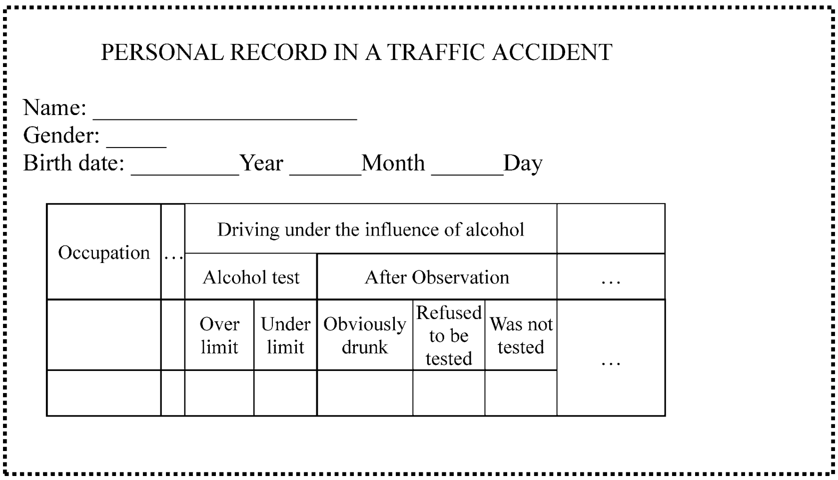
A form can be decomposed into certificating, extensional, intensional and descriptive parts [1]. The certificating part contains data related to the formality of the form, such as data of issuance, stamps, marks, identifier, serial number or signatures. The extensional part contains cells that must be filled in with values. The intensional part contains headings that indicate the meanings of the values in different cells. The descriptive part contains instructions or rules for filling out the extensional part. The intensional and extensional parts are often highly interconnected. In fact, the intensional part contains indices of the extensional part. A combination of the intensional and extensional parts is called an area.

The structure of a form may be analyzed by its entities, relationships and associated attributes. In Fig. 1, for instance, the personal record in a traffic accident involves entities, such as person, vehicle and accident. Entities can have relationships. For instance, "A person drives a vehicle" and "Vehicles have accidents" are relationships. Both entities and relationships may have attributes. A person may have name, birth date, gender and education as its attributes. Accident may have time, place, damage and other attributes. Attributes may be simple or composite. An attribute with no subattribute is called a *simple attribute* (or elementary property), whereas an attribute having other attributes as its subattributes is called a *composite attribute*. For example, gender is a simple attribute, but birth date is a composite one, because the birth date has day, month and year as its subattributes.

Each attribute is defined on a particular domain and can have values. For example, age may be defined on a domain of 0–200 years and gender is defined as either male or female. Depending upon the nature of their domains, attributes may be nominal or nonnominal. A nominal attribute has categorical values, whereas a nonnominal attribute has numerical values. For example, gender is nominal because its value is either male or female, while age is nonnominal. If age is divided into several categories (such as adult, child and infant) instead of using the actual year and month, then it becomes nominal.

An attribute of interest to the user is a field. A form is a collection of related fields that are organized to meet the user's information needs. When designing a form, fields may be grouped by certain similarities among attributes. One way is to group fields based on whether they allow multiple values in a single form. For example, gender is single-valued in a form of personal record, but education can be multiple-valued, which includes data from primary school to the highest degree. In Fig. 1, the single-value attributes are presented in separate lines, whereas the multiple-value attributes, including occupation, education, vehicle type and driver's license information, are grouped into a table. Data in a form can be displayed in parametric texts or in tables. Parametric texts are a set of texts with fields to be filled in with proper values. For example, "we certify that Mr. _____ was born on _____," is a set of parametric texts containing two fields.

A table includes headings and cells. The headings are indices or classifiers of the value in the cell. In a two-dimensional form, row and column headings show the meaning of row and column attributes, respectively. Fig. 2 is a sample multi-dimensional form, with region as its row headings and a combination of the driver's alcohol test and gender as column headings. A cell is a repository of attribute



PERSONAL RECORD IN A TRAFFIC ACCIDENT

Name: ______________________

Gender: _____

Birth date: _________Year ______Month ______Day

Occupation	…	Driving under the influence of alcohol					
		Alcohol test		After Observation			…
		Over limit	Under limit	Obviously drunk	Refused to be tested	Was not tested	…

Fig. 1. Part of a personal record in a traffic accident report.

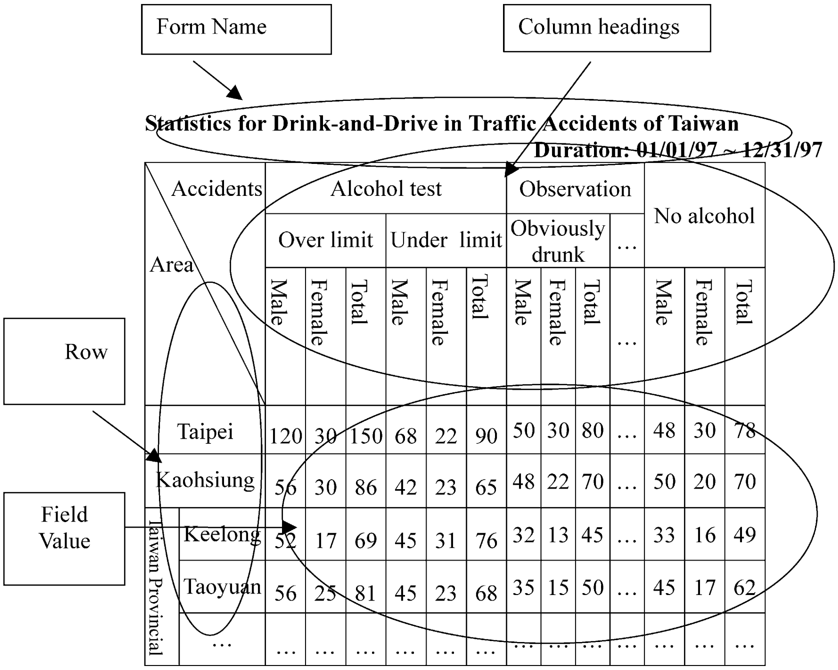


Fig. 2. A two-dimensional form.

values. It may be filled with numbers or mathematical functions that can generate values.

Headings may be simple or composite. A simple heading has a simple attribute. For example, if gender is used as a column heading in a tabular form, it is a simple heading with two categories, male and female. A composite heading involves composite attributes that may be a single composite attribute, such as the row headings (region) in Fig. 2, or a combination of more than one simple attribute, such as the column headings (a combination of alcohol test and gender) in Fig. 2. Headings may be classified into different types. For example, we may define *Accident items* as a type that represents the relationship of attributes associated with car accidents in a hierarchy. For instance, the column heading of Fig. 2 is an element of *Accident items*. Types are useful for generalizing headings.

3. Levels of abstraction

In addition to grouping related attributes into a table, forms can also be generalized at three different levels: form instance, form template and meta-template.

(1) Form instance: a special instance of a form template, with all cells being instantiated with proper values. It is the most common form we see. In this research, we use "form" and "form instance" interchangeably.
(2) Form template: the skeleton of a form, in which attribute values are removed and may be substituted by other proper ones. The example in Fig. 3 shows the relationship between a form and its template. A template may contain headings, fixed texts, operation expressions, graphics and other fields that are not stored in the database.
(3) Meta-template: a further abstraction of templates by replacing their headings with associated types. For instance, the column and row headings of the form templates shown in Fig. 3 can be generalized into a meta-template of *Accident items* because the attributes (including vehicle type, education, casualty and vehicle motion) belong to the types of *Accident items*. A meta-template can generate

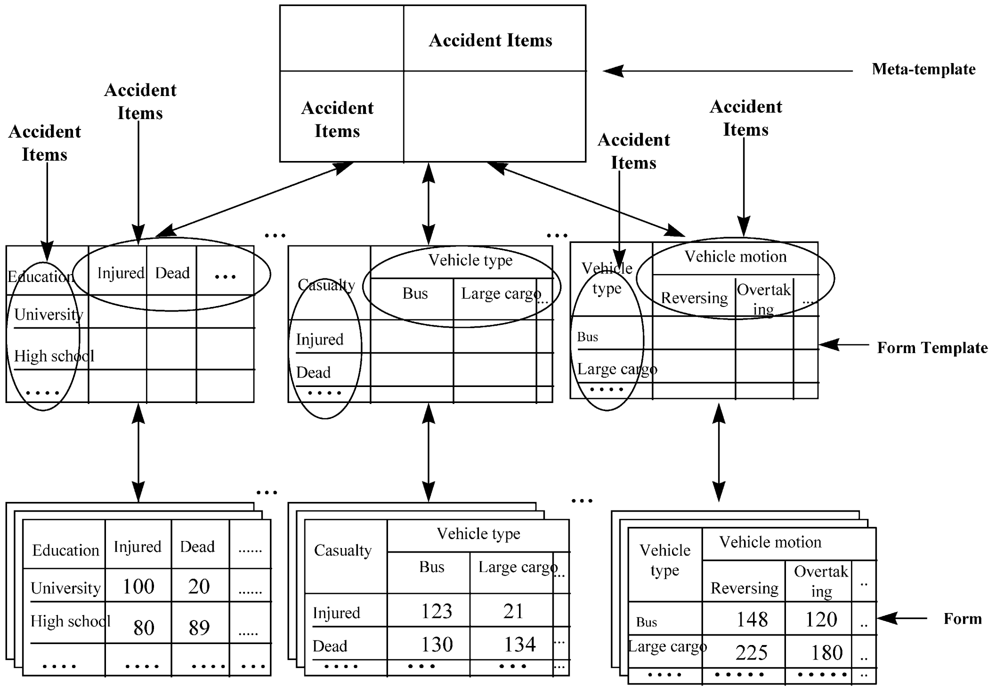


Fig. 3. Meta-template, template, form and their relationship.

multiple templates. A template can generate multiple forms.

4. Factoring and synthesis of forms

Based on different levels of abstraction, forms can be manipulated and managed through a factoring and synthesis process, as shown in Fig. 4 (adapted from Ref. [7]). Factoring is a process of aggregation and generalization. It builds templates and meta-templates from existing forms. Synthesis is a process of specialization and instantiation. It constructs forms from meta-templates and templates.

The first step in factoring is form analysis that extracts form headings to build a template. Cell values in business forms are removed from tables to separate templates and their affiliated data (F1). The templates are then generalized into meta-templates (F2). A structure can be built from the headings. Data are stored and indexed by the heading structure in a database for efficient retrieval in the future.

Construction of meta-templates is based on type similarities among headings. In Fig. 4, for instance, the column and row headings of the left-most template are the causes of accidents and the regions where accidents occurred. The column and row headings of the second template from the left are the driver's education and the regions where the accident occurred. Because both row headings of these two templates are geographic regions used for classification and the column headings are items used for observation and analysis of accidents, they share the same meta-template. The column headings are of *Accident items* and row headings are of geographical *Region*. Here, *Accident items* and *Region* are two types. A meta-template of *Region* and *Accident items* can be defined.

The synthesis process constructs forms from meta-templates and templates. When a form is needed, the user chooses a proper meta-template to build a template by defining the row and column headings (S1). Once the template is built, the system retrieves data from the database and maps data into the template based on the specification of the template. A form is constructed (S2).

In the actual application, the synthesis process includes two stages: template definition and informa-

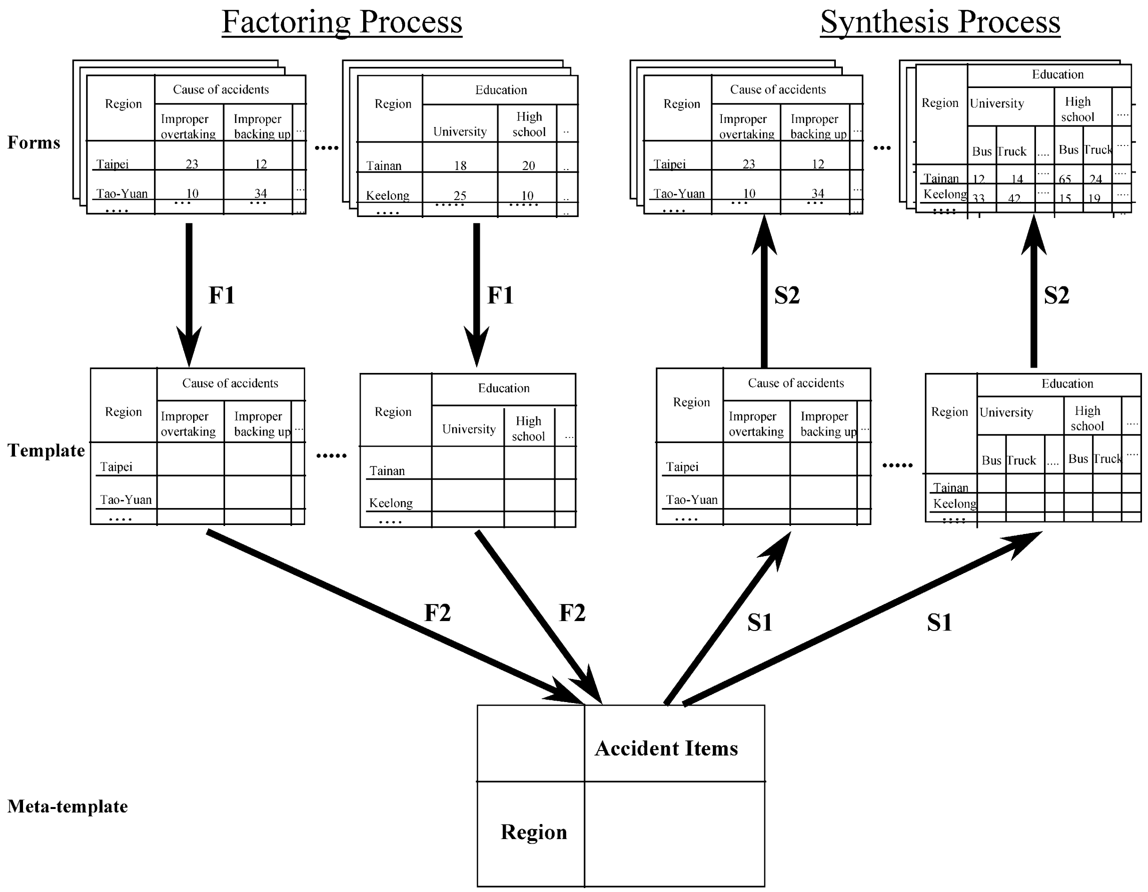


Fig. 4. Factoring and synthesis of forms.

tion display (see Fig. 5). At the template definition stage, the user chooses a meta-template and then defines row and column headings to build a preliminary template from the requirement. The preliminary template may be edited to meet the exact information needs, if necessary. At the information display stage, the system generates SQL queries to retrieve relevant data from the database and then fill in the template properly for information presentation.

5. Methodology for designing form-based DSS

The processes of factoring and synthesis suggest a method for designing form-based DSS. The method includes two major stages: design and application (see Fig. 6). The design stage is the process of factoring that includes form analysis, heading structure design, meta-template design and database design. Once the meta-template and database are available, the system can be applied to construct forms by the synthesis process. This section presents major modules of the method and corresponding algorithms.

5.1. Form analysis

The objective of form analysis is to find major attributes for designing the DSS. Each field in the form needs to be examined carefully to identify and separate different parts and areas. Given a set of business forms, the first step is to decompose them into their structure and affiliated data. For each form, a data glossary is produced to describe the attributes (i.e., fields) contained in the form, their elementary

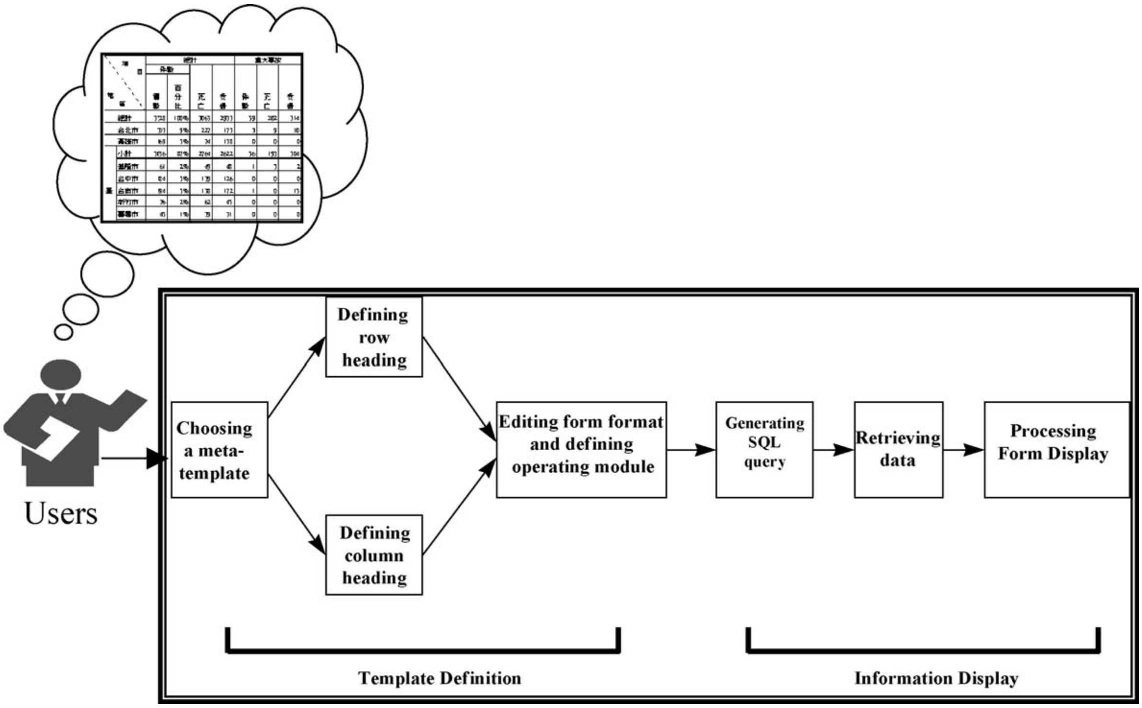


Fig. 5. Processing procedure of the flexible form presentation.

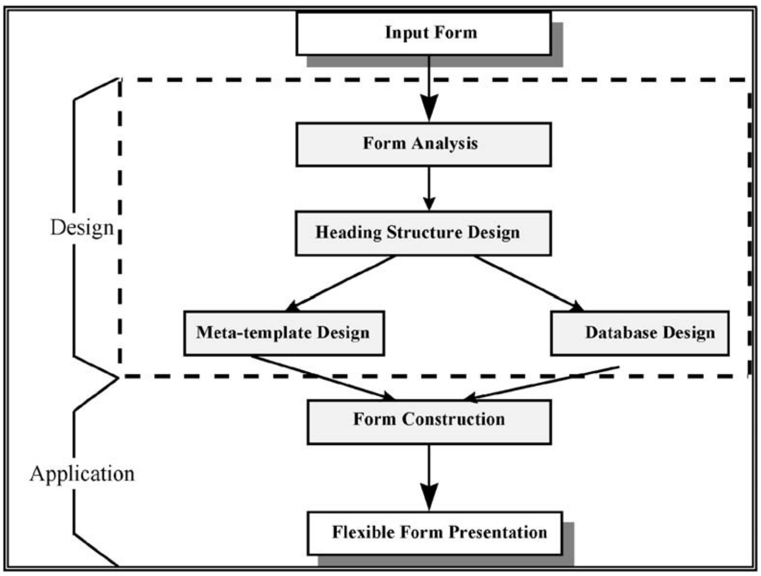


Fig. 6. Methodology for development.

properties and abstraction hierarchies. The process of form analysis is recursive, as described below:

Begin
  For each form
    (1) Separate different parts (descriptive parts, extensional/intensional parts)
    (2) Identify areas and subareas, assign a name to each of them
    (3) For each area, repeat
      (3.1) Extract entity
      (3.2) Analyze the attributes of the entity
      (3.3) Fill data glossary with attributes and build attribute hierarchies
      (3.4) Save attributes, entities and attribute hierarchies.
      Until all areas are examined.
  Until all forms are examined
  Merge the data glossary and resolve conflicts
End of form analysis

To elaborate the procedures, forms are analyzed to identify their parts (descriptive parts and extensional/intensional parts), areas and attributes. We use the form in Fig. 7 as an example. It has parametric text and tables. The extensional/intensional part of the form has the following fields in the parametric text: "date of accident", "place of accident", "casualty", "number of fast lanes" and "speed limit." The table has two main areas: road conditions (area 1.1) and traffic facilities (area 1.2).

The second step focuses on defining areas in the form. The process of identifying areas is recursive. The fields describing the same type of concept should be grouped into one area. It is common that the fields in a hierarchical structure have common properties. Each area must have a name and an identification number. In Fig. 7, the field, *traffic facilities*, is a table consisting of a group of subareas, such as "traffic lights", "road division" and "traffic signs". If we consider the whole form as an area, label it as *Area1*

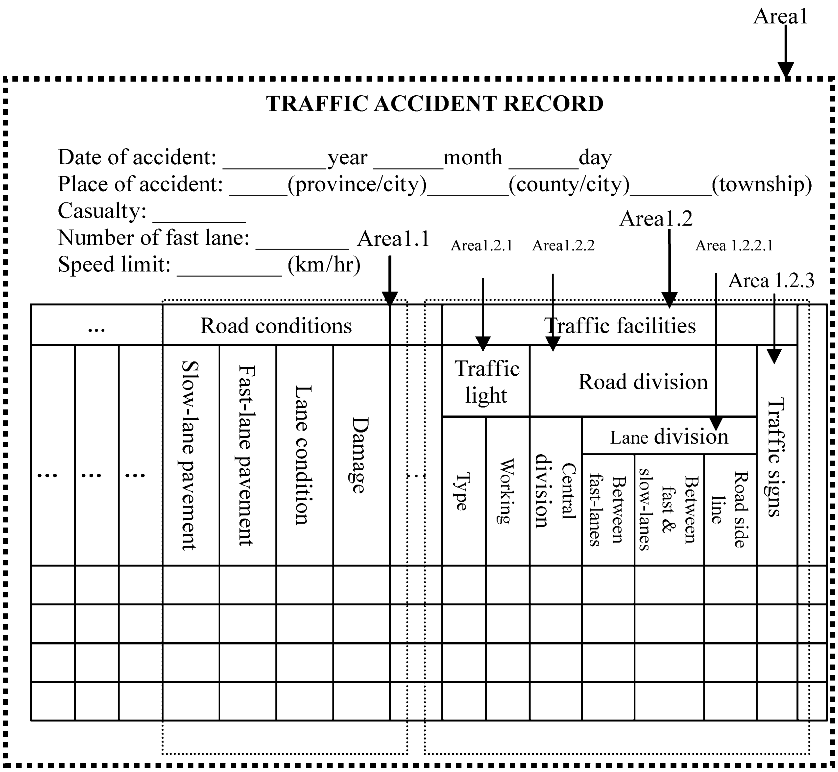


Fig. 7. Traffic accident record (partial).

and name it as *traffic accident*. *Area1* can be further divided into several subareas, labeled as *Area1.1* (named *road conditions*) and *Area1.2* (named *traffic facilities*), respectively.

The third step is to elicit attributes from the form. To do so, we need to extract concepts from the identified parts and areas. As shown in Fig. 7, the parametric text includes attributes, such as *date of accident* or *place of accident*. In the table, the field names are attributes, such as *road conditions* and *traffic facilities*. Next, we check whether an attribute is simple or composite. In general, the attribute extracted from the parametric text can be further decomposed. For instance, the attribute, *date of accident,* in Fig. 7 is normally represented as year–month–day. It may be further decomposed into three attributes: *year*, *month* and *day*. This decomposition allows data to be analyzed in more detail.

To describe the resulting attributes, a data glossary of all attributes must be built. It includes code, name, areas, description, instances and synonyms as shown in Table 2. Description, instances and synonyms are useful in having a clear understanding of the meaning and role of each attribute in an application. Normally, a system has more than one form. Therefore, data glossaries obtained from different forms are integrated to derive a complete data glossary. During the process of integration, restructuring is often needed. Attention must also be paid to handle naming conflicts and redundancy in areas and attributes.

5.2. Heading structure design

After defining all attributes, it is necessary to find their relationships. The resulting structure can be used as headings for constructing forms and indices for organizing data for efficient storage and retrieval. In this step, similar items are grouped by aggregation or generalization to build hierarchies. The procedure of heading structure design is shown as follows. The algorithm that constructs hierarchies automatically is shown in Appendix A.

```
Heading structure design
Begin
  Initialize a data group
  For each form
    For each item on the column heading and
    row heading Do
      If the item can be classified into one of the
      data groups
      Then add it to the data group and
      construct the hierarchical relationship
      among the items
      Else build a distinct data group
    Until all items are examined
  Until all forms are examined
End of heading structure design
```

In designing the heading structure, aggregation and generalization are utilized to identify hierarchical relationships within each data group. Aggregation allows several attributes of different nature to be combined into a higher-level concept. In Fig. 8, the concepts, such as *slow-lane pavement*, *fast-lane pavement*, *lane condition* and *damage* can be aggregated into a higher-level concept, called *road conditions*. The aggregation of several attributes creates a composite attribute.

Generalization allows several concepts of similar nature to be represented by a higher-level concept. As shown in Fig. 9, the concepts of *bus*, *cargo truck*, *truck* and *full-size trailer* are similar in that they are vehicles and, hence, can be generalized to a higher-level concept, called *vehicle type*. The results of aggregation and generalization may be recorded under "aggregation of" and "generalization of" in the data glossary (see Table 2).

In the DTTG project, we applied aggregation and generalization to construct the hierarchical structure by analyzing and integrating the attributes resulted from form analysis. Three hierarchical structures are constructed after investigating the data glossaries and interviewing end users: *time*, *region* and *accident items*. *Time* indicates when the accident occurred. *Region* indicates the geographic areas in Taiwan. The *Accident items* show the circumstances and facilities associated with the accident, such as the weather conditions, the traffic facilities and road conditions.

Figs. 10 and 11 are the hierarchical structures of *Region* and *Accident items*. *Region* has four levels. The first level is *Taiwan Island*, followed by *province/city*, *county* and *township*. The structure of *Accident*

Table 2
Partial data glossary of Fig. 2

Form: personal record in traffic accident

Code	Name	Concept description	Instance	Synonyms	Aggregation of:	Generalization of:	Par.	T
D001	Name	Name of the person involved	John					T1
D002	Gender	Gender of the person involved	Male					T2
D003	Birth date	Birth date of the person involved	1968/07/23		Year, month, day			T3
D004	Year	Year of the birth date	1998				D003	T3
D005	Month	Month of the birth date	07				D003	T3
D006	Day	Day of the birth date	23				D003	T3
D007	Occupation	Occupation of the person involved	Teacher					T4
D008	Driving under the influence of alcohol	Test drive under the influence of alcohol				Alcohol test, after observation		T5
D009	Alcohol test	Result of the alcoholic test	Over limit, under limit				D008	T5
D010	After observation	Driver's observed alcoholic condition	Obviously drunk, refused to be tested, not tested				D008	T5
...	...	...	...	...	...	...	...	
...	...	...	...	...	...	...	...	
D301	Accident date	Date of accident	1998/11/12		Year, month, day			T11
D302	Year	Year of the accident date	1998				D301	T11
D303	Month	Month of the accident date	11				D301	T11
D304	Day	Day of the accident date	22				D301	T11
D305	Accident place	The place where an accident occurred			Province/city, county/city, township			T12
D306	Province/city	The province/city of the accident place	Taiwan, Taipei				D305	T12
D307	County/city	The country/city of the accident place	Pingtung				D305	T12
D308	Township	The township of the accident place	Wantan				D305	T12
D309	No. of fast lane	No. of fast lane at the accident place	2					T13
D310	Speed limit	Speed limit at the accident place	96 km					T14
...	...	...	...	...	...	...	...	...
D312	Road conditions	Road types and conditions at the accident place			Slow-lane pavement, fast-lane pavement, lane condition, damage			T15
D313	Slow-lane pavement	The condition of the slow lane at the accident place	Paved, not paved				D312	T15

Table 2 (*continued*)

Form: personal record in traffic accident

Code	Name	Concept description	Instance	Synonyms	Aggregation of:	Generalization of:	Par.	T
D314	Fast-lane pavement	The condition of the fast lane at the accident place	Paved with asphalt, paved with semen				D312	T15
	...	...	...	...	...	...	...	
D317	Traffic facilities	The traffic facilities around the accident place			Traffic light, road division, traffic signs			T21
D318	Traffic light	The traffic light at the accident place				Type, working	D317	T21
D319	Road division	The road division at the accident place			Central, lane		D317	T21
D320	Lane division	The lane division at the accident place				Between fast lanes, between fast and slow lanes, road side line	D317	T21
D321	Traffic signs	The traffic signs at the accident place					D317	T21
...	...	...	...	...	...	...	...	

items has seven levels. For instance, *no passing* is an attribute at the seventh level. If we trace up from this attribute, the attributes at six different levels are *multiple fast lanes*, *between fast lanes*, *lane division*, *road division*, *traffic facilities* and *traffic accident*, respectively.

5.3. Meta-template design

The heading structures constructed previously serve as the basis for designing meta-templates and databases. The purpose of meta-template design is to simplify and automate the process of form generation. The procedure is shown as follows. An algorithm for automating the meta-template construction is described in Appendix B.

Begin
  Initialize a meta-template
  For each template or form
    (1) Build meta-templates by replacing the column and row headings with the name of their hierarchical structures
    (2) If the meta-template already exists, drop the redundant one. Otherwise, save the new meta-template
  Until all forms or templates are examined
End of meta-template design

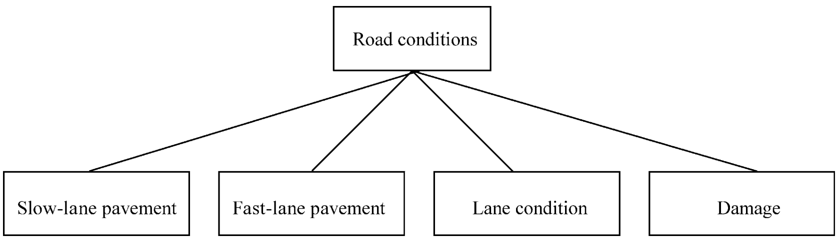


Fig. 8. An example of aggregation.

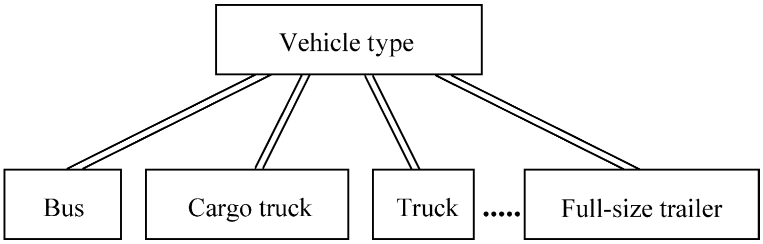


Fig. 9. An example of generalization.

The column and row headings of the left-most template in Fig. 4 are the causes of accidents and the regions where the accidents occurred, respectively. Similarly, the column and row headings of the second template from the left are drivers' educational levels and the regions where the accidents occurred. Because both row headings of these two templates belong to the type of *Region* and both column headings belong to the type of *Accident items*, the meta-template has *Accident items* as its column heading and *Region* as its row heading. Fig. 12 shows four meta-templates defined in the DTTG project from the forms used by the government agency: *Region/Accident items*, *Accident items/Accident items*, *Time/Accident items* and *Region_Accident items/Accident items*.

5.4. Database design

To maximize the flexibility in form construction, data must be decomposed and stored at the elementary level. To accomplish this goal, data are indexed by the heading structures built in the previous step. For instance, if there exist n distinct types of heading structures (data type 1...n), we can define the main database schema as {data type1, data type2, ..., data type_n, data value}. The data length of each attribute in the above schema is the maximum number of layers in the heading structure.

In our prototype implementation, there are three heading structures: *Time, Region* and *Accident items*. Thus, the main database schema can be represented as follows: *Time*, *Region*, *Accident items*, *value*. The number of layers for the heading structure of *Time*, *Region*, and *Accident items* are two, four and seven, respectively. Assuming that each layer can be represented in two decimal digits and *Value* can be represented in eight decimal digits, the data lengths of the attributes will be 4, 8, 14 and 8, respectively. Because the database design is not the focus of this research, details of schema design and normalization are not discussed.

5.5. Form construction

The major function of the application phase is to construct forms from meta-templates and templates. To allow end users to create their own forms easily, it is necessary to automate the application process.

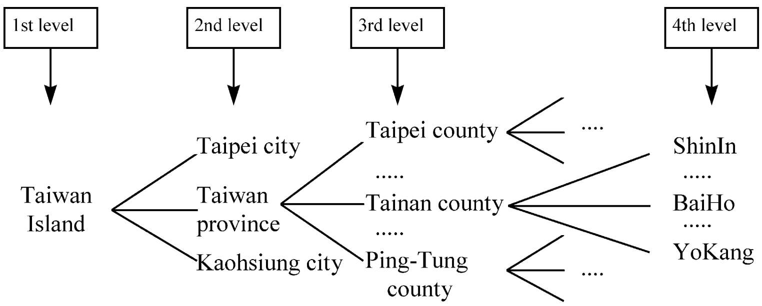


Fig. 10. The hierarchical structure of region.

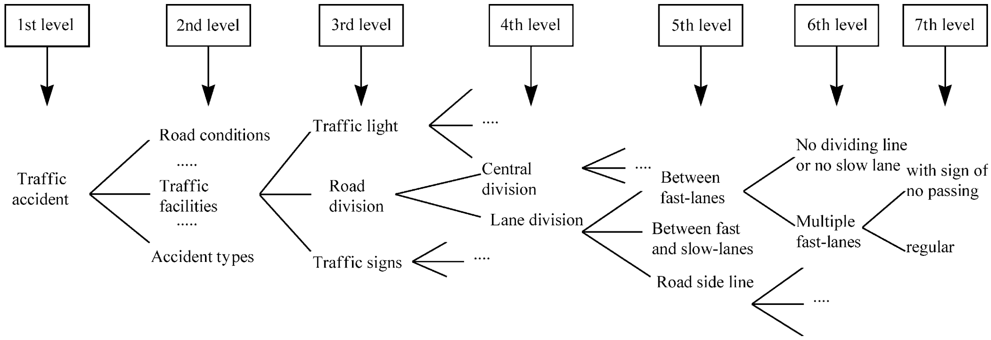


Fig. 11. The hierarchy of accident items.

The following are procedures for automatic form construction.

(1) Template construction
  (1.1) Select a suitable meta-template
  (1.2) Specify the column and row headings and then display them in a spreadsheet environment
  (1.3) Define the mathematical functions in the cell and edit the layout, if necessary
(2) Form display
  (2.1) Build SQL queries based on the chosen template and execute the queries to retrieve the data
  (2.2) Maps the resulting data into the template and performs necessary operations as specified in the template

In our implementation, users should determine what data they need and how the data should be displayed in order to choose a suitable meta-template in the template construction phase. They then specify the row and column headings using the structures constructed in the phase of heading structure design. The selected items will be transmitted to an editing area for further operational specification and template editing. When users perform form display, the system automatically generates the necessary SQL queries based on the constructed template, performs query execution and maps the resulting data to the template. The resulting form can be either viewed on the screen or printed from the printer.

6. Flexform: a prototype implementation

To demonstrate the feasibility of the proposed methodology, a prototype system, called Flexform, has been developed. Flexform was implemented in an environment that integrates a relational database management system (RDBMS) and spreadsheet model

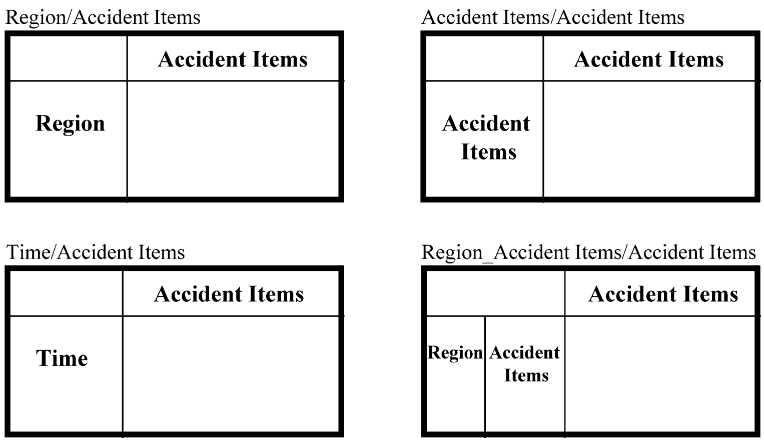


Fig. 12. Four templates in the DTTG project.

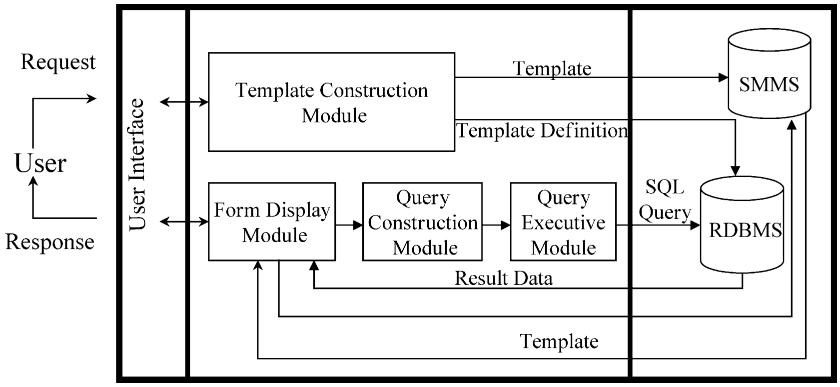


Fig. 13. The architecture of Flexform.

management system (SMMS). The development tools used were Microsoft Excel, Delphi and Microsoft SQL Server. Fig. 13 depicts Flexform's architecture, which contains four modules: template construction, query construction, query execution and form display. The template construction module is used to create templates from meta-templates. The query construction and query execution modules help construct the needed SQL queries and retrieve data from RDBMS. The form display module combines the template and data to build forms.

To show how the user applies the system to generate forms, the example of traffic accident analysis described previously was implemented. In the example, Flexform has three distinct heading structures: *Time*, *Region* and *Accident items*. In addition, the system has four meta-templates: *Region/Accident items*, *Accident items/Accident items*, *Time/Accident items* and *Region_Accident items/Accident items*. They are shown in the left-top, right-top, left-bottom and right-bottom corners in Fig. 14, respectively.

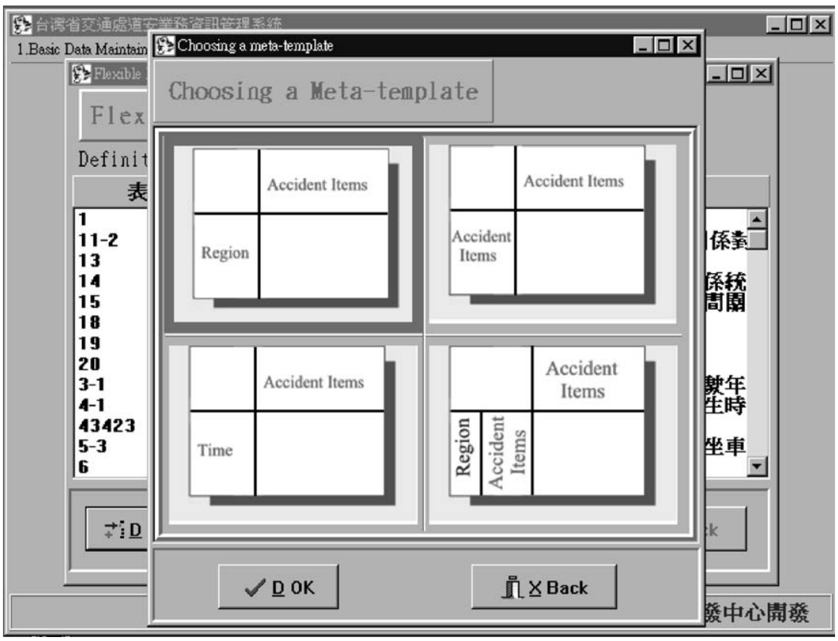


Fig. 14. Meta-template of Flexform system.

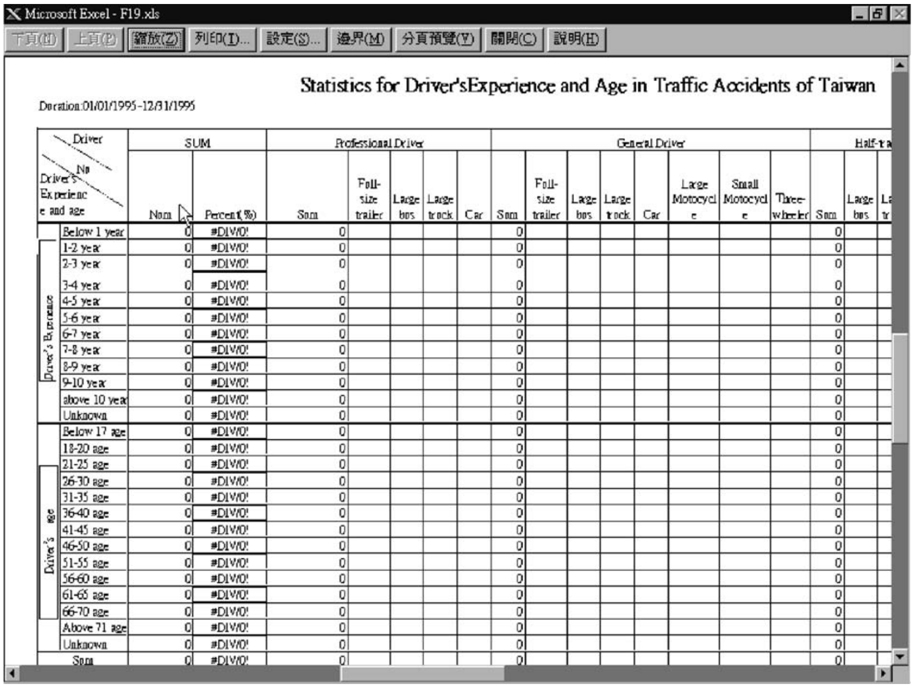
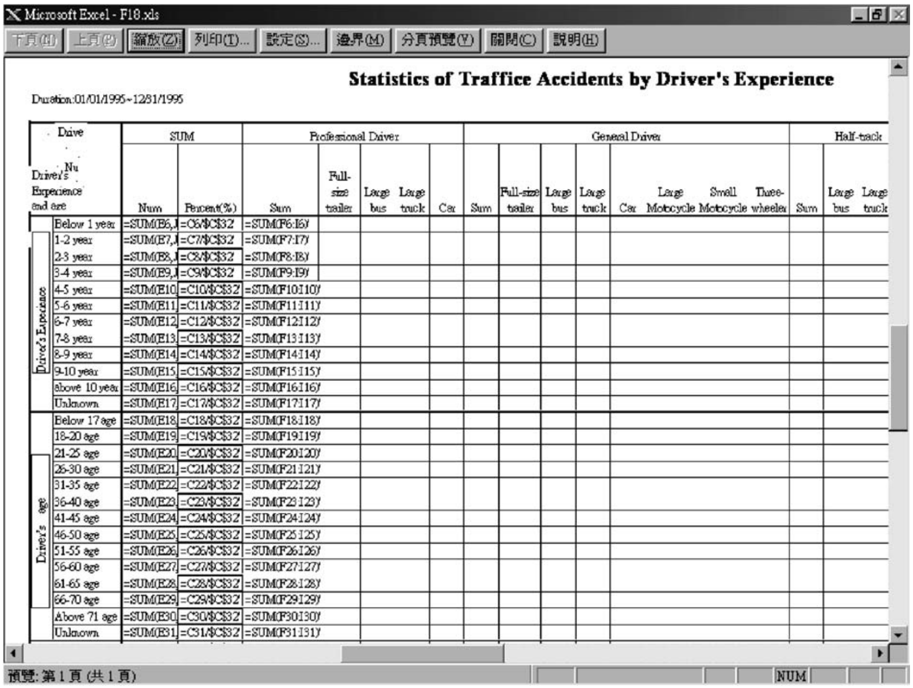


Fig. 15. The desired form.

Fig. 16. Specifying column items.

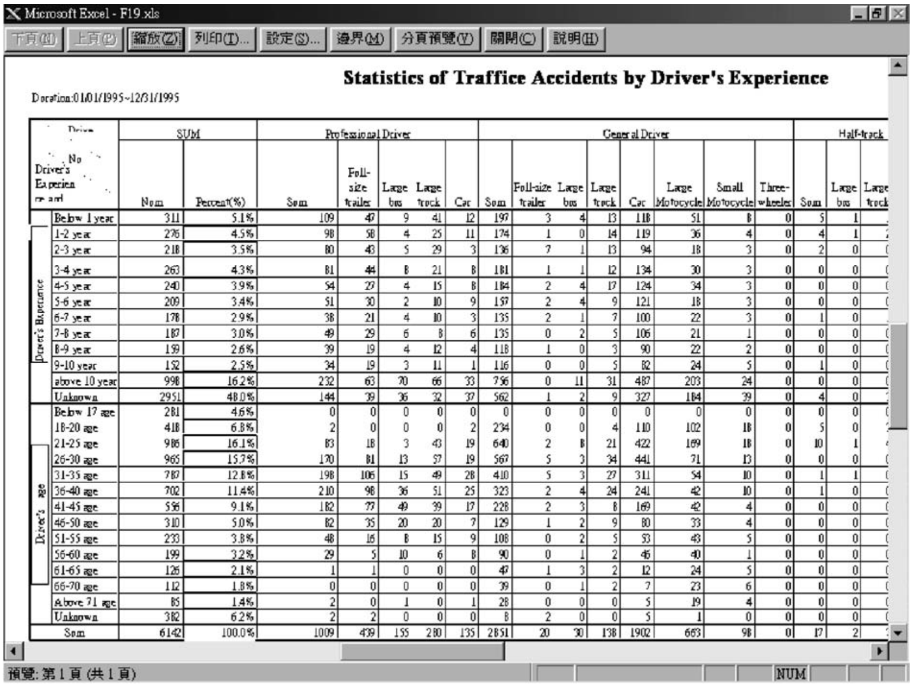


Statistics of Traffice Accidents by Driver's Experience

Duration:01/01/1995~12/31/1995

Driver's Experience and age \ Drive Nu		SUM: Num	SUM: Percent(%)	Professional Driver: Sum	Full-size trailer	Large bus	Large truck	Car	General Driver: Sum	Full-size trailer	Large bus	Large truck	Car	Large Motocycle	Small Motocycle	Three-wheeler	Half-track: Sum	Large bus	Large truck
	Below 1 year	=SUM(E6,I	=C6/$C$32	=SUM(F6:I6)															
Driver's Experience	1-2 year	=SUM(E7,I	=C7/$C$32	=SUM(F7:I7)															
	2-3 year	=SUM(E8,I	=C8/$C$32	=SUM(F8:I8)															
	3-4 year	=SUM(E9,I	=C9/$C$32	=SUM(F9:I9)															
	4-5 year	=SUM(E10	=C10/$C$32	=SUM(F10:I10)															
	5-6 year	=SUM(E11	=C11/$C$32	=SUM(F11:I11)															
	6-7 year	=SUM(E12	=C12/$C$32	=SUM(F12:I12)															
	7-8 year	=SUM(E13	=C13/$C$32	=SUM(F13:I13)															
	8-9 year	=SUM(E14	=C14/$C$32	=SUM(F14:I14)															
	9-10 year	=SUM(E15	=C15/$C$32	=SUM(F15:I15)															
	above 10 year	=SUM(E16	=C16/$C$32	=SUM(F16:I16)															
	Unknown	=SUM(E17	=C17/$C$32	=SUM(F17:I17)															
	Below 17 age	=SUM(E18	=C18/$C$32	=SUM(F18:I18)															
	18-20 age	=SUM(E19	=C19/$C$32	=SUM(F19:I19)															
Driver's age	21-25 age	=SUM(E20	=C20/$C$32	=SUM(F20:I20)															
	26-30 age	=SUM(E21	=C21/$C$32	=SUM(F21:I21)															
	31-35 age	=SUM(E22	=C22/$C$32	=SUM(F22:I22)															
	36-40 age	=SUM(E23	=C23/$C$32	=SUM(F23:I23)															
	41-45 age	=SUM(E24	=C24/$C$32	=SUM(F24:I24)															
	46-50 age	=SUM(E25	=C25/$C$32	=SUM(F25:I25)															
	51-55 age	=SUM(E26	=C26/$C$32	=SUM(F26:I26)															
	56-60 age	=SUM(E27	=C27/$C$32	=SUM(F27:I27)															
	61-65 age	=SUM(E28	=C28/$C$32	=SUM(F28:I28)															
	66-70 age	=SUM(E29	=C29/$C$32	=SUM(F29:I29)															
	Above 71 age	=SUM(E30	=C30/$C$32	=SUM(F30:I30)															
	Unknown	=SUM(E31	=C31/$C$32	=SUM(F31:I31)															

Fig. 17. Construction of the template.



Statistics of Traffice Accidents by Driver's Experience

Duration:01/01/1995~12/31/1995

Driver's Experience and age \ Drive No		SUM: Num	SUM: Percent(%)	Professional Driver: Sum	Full-size trailer	Large bus	Large truck	Car	General Driver: Sum	Full-size trailer	Large bus	Large truck	Car	Large Motocycle	Small Motocycle	Three-wheeler	Half-track: Sum	Large bus	Large truck
	Below 1 year	311	5.1%	109	47	9	41	12	197	3	4	13	118	51	8	0	5	1	
Driver's Experience	1-2 year	276	4.5%	98	58	4	25	11	174	1	0	14	119	36	4	0	4	1	
	2-3 year	218	3.5%	80	43	5	29	3	136	7	1	13	94	18	3	0	2	0	
	3-4 year	263	4.3%	81	44	8	21	8	181	1	1	12	134	30	3	0	0	0	
	4-5 year	240	3.9%	54	27	4	15	8	184	2	4	17	124	34	3	0	0	0	
	5-6 year	209	3.4%	51	30	2	10	9	157	2	4	9	121	18	3	0	0	0	
	6-7 year	178	2.9%	38	21	4	10	3	135	2	1	7	100	22	3	0	1	0	
	7-8 year	187	3.0%	49	29	6	8	6	135	0	2	5	106	21	1	0	0	0	
	8-9 year	159	2.6%	39	19	4	12	4	118	1	0	3	90	22	2	0	0	0	
	9-10 year	152	2.5%	34	19	3	11	1	116	0	0	5	82	24	5	0	1	0	
	above 10 year	998	16.2%	232	63	70	66	33	756	0	11	31	487	203	24	0	0	0	
	Unknown	2951	48.0%	144	39	36	32	37	562	1	2	9	327	184	39	0	4	0	
	Below 17 age	281	4.6%	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	18-20 age	418	6.8%	2	0	0	0	2	234	0	0	4	110	102	18	0	5	0	
Driver's age	21-25 age	986	16.1%	83	18	3	43	19	640	2	8	21	422	169	18	0	10	1	
	26-30 age	965	15.7%	170	81	13	57	19	567	5	3	34	441	71	13	0	0	0	
	31-35 age	787	12.8%	198	106	15	49	28	410	5	3	27	311	54	10	0	1	1	
	36-40 age	702	11.4%	210	98	36	51	25	323	2	4	24	241	42	10	0	1	0	
	41-45 age	556	9.1%	182	77	49	39	17	228	2	3	8	169	42	4	0	0	0	
	46-50 age	310	5.0%	82	35	20	20	7	129	1	2	9	80	33	4	0	0	0	
	51-55 age	233	3.8%	48	16	8	15	9	108	0	2	5	53	43	5	0	0	0	
	56-60 age	199	3.2%	29	5	10	6	8	90	0	1	2	46	40	1	0	0	0	
	61-65 age	126	2.1%	1	1	0	0	0	47	1	3	2	12	24	5	0	0	0	
	66-70 age	112	1.8%	0	0	0	0	0	39	0	1	2	7	23	6	0	0	0	
	Above 71 age	85	1.4%	2	0	1	0	1	28	0	0	0	5	19	4	0	0	0	
	Unknown	382	6.2%	2	2	0	0	0	8	2	0	0	5	1	0	0	0	0	
Sum		6142	100.0%	1009	439	155	280	135	2851	20	30	138	1902	663	98	0	17	2	

Fig. 18. Form constructed by Flexform.

Suppose that the user needs to generate a form that compares the frequency of accidents by the driver's age and experience, as shown in Fig. 15. The user may choose the meta-template located on the right-top of Fig. 14 to start with.

Once a meta-template is chosen, the system pops up a window (Fig. 16) that allows the user to specify row and column headings of the template. After a column heading is chosen (i.e., *Accident items* in this case), the system pops up a new window with a data structure containing the *Accident items*. The user can use the mouse to mark and move the needed items from the source window (the left-hand side window in Fig. 16) to the destination window (the right-hand side window in Fig. 16). A similar procedure can be used to specify row headings.

The template construction module records the row and column items chosen by the user and sends them to the editing area for possible refinement. Users may polish the format of the preliminary template and save the final version (see Fig. 17). Following the template construction is the form display function that retrieves data from the database and fills into the template. First, the query construction module is triggered to generate SQL queries. Then, Flexform executes SQL, performs the necessary cross-referencing and calculations to retrieve appropriate data, and then saves them into a temporary file. Finally, the form display module generates another set of SQL queries to feed the data from the temporary file into the template. The functions defined in the template are automatically executed. That is, only the result is shown in the final form. For instance, fields whose values must be derived from the values of other fields are calculated automatically. Fig. 18 shows the resulting form built by flexform.

7. Conclusions

The paper presents a methodology for designing form-based DSS. The method is based on the concept of factoring and synthesis to simplify form management. It includes two major phases: one is to derive meta-templates and design databases from existing forms; the other applies meta-templates to create new forms for application. A prototype system has also been implemented to demonstrate the feasibility of the method.

The contribution of the paper is three-fold. First, the methodology can alleviate the difficulty in flexible presentation of numerical data, especially when frequent change is necessary. It integrates several concepts and methods (such as factoring, synthesis, aggregation and generalization) into the design process to provide flexibility necessary for end users to create their own forms easily. Second, the method allows a form skeleton and its associated data to be managed separately. This helps the construction of form-based DSS to better use existing data in the database. The user may specify different templates and fill them with data in the current database. It also provides a greater possibility for reusing tables and other presentation modules. Finally, the method intends to integrate semantic knowledge to support form generation. This makes it different from existing methods and has the potential of greatly improving the capabilities of ad hoc analysis and on-line analytic processing in DSS.

This work is the beginning of a line of research focused on flexible form management in DSS. Future research directions are abundant. For instance, in order to increase the applicability of flexible forms, methods must be developed to allow forms to be compatible with different data models and different database systems. This may include integration of models and solvers to manage form models, as mentioned in Ref. [6]. Other issues include integration with other systems to expand system capabilities and evaluation of system productivity and user satisfaction. Applications to improve existing report generation functions, such as the *Report Wizard* in Microsoft Access, will also have significant value.

Acknowledgements

We would like to thank anonymous referees for their helpful comments. This research was partially supported by the National Science Council of Taiwan under the grant NSC87-2416-H-110-023. Many concepts embodied in this work were initially kindled by a decision support system project from the Taiwan Provincial Government. The implementation of the system was carried out at the Software Research and Development Center (SR and DC), College of Management, National Sun Yat-Sen University.

Appendix A. Algorithm for automatic construction of hierarchies

```
/* This algorithm is designed to construct the hierarchical relationships among the
attributes in the data glossary.  Similar items are grouped following the aggregation
and generalization rules.  Its input and output are as follows:
   Input: data glossary (DG).  Each data item in the DG has four fields: item code
          (Code), item name (Name), parent node (Parent), and tree structure (Tree).
   Output: tree structures that show the hierarchical relationships.
*/
Begin
Input data glossary (DG)
/* Variable i is a temporary index */
i = 1;
/* Find the data item whose parent field is Null, create a new
  tree, and assign the data item to be the root */
For m = 1 to total number of DG
  If DG[m].parent = Null
    /* create a new node */
    New(T&i);
    /* assign T[i] to be the root of the tree */
    T&i.Tree = "T"+&i;      /* generate a string such as "T1", "T2." */
    T&i.Code = DG[m].Code;
    T&i.Name = DG[m].Name;
    T&i.Parent = Null;
    i = i + 1;
  Endif;
Next m;
/* Link all data items in the DG to their parent nodes */
For m = 1 to total number of DG
  IF DG[m].Parent <> Null and DG[m].Tree = Null
    /* create a new node */
    New(NewNode);
    NewNode.Code = DG[m].Code;
    NewNode.Name = DG[m].Name;
    NewNode.Parent = DG[m].Parent;
    i = 1;
    /* Find the data item and link it to the related tree */
    Do
      NewNode.Tree = DG[i].Tree;
      i = i+1;
    Until DG[m].Parent = DG[i].Code;
  Endif;
Next m;
End;
```

Appendix B. Algorithm for meta-template construction

```
/* This algorithm is designed to construct the meta-template from a form.
  Input: the level-sequence index of a form (FLS)
  Output: meta-template.
  Variable definitions:
  ColLvl: column level
  ColSeq: column sequence at each column level
  MaxColLvl: maximum column level
  MaxColSeq: maximum column sequence
  RowLvl: row level
  RowSeq: row sequence at each row level
  MaxRowLvl: maximum row level
  MaxRowSeq: maximum row sequence
  Structure: data_type
    Data: data item
    Tree: The tree to which a data item belongs, such as "T1", "T2"
  Col_meta-template: Array[1..n, 1..MaxColLvl, 1..MaxColSeq] of data_type
  Row_meta-template: Array[1..n, 1..MaxRowLvl, 1..MaxRowSeq] of data_type
  ColItem, RowItem: data_type, variables that indicate the particular column
                    and row of a data item in a FSL, respectively
  Structure: meta_type
    ColData: the data item at the column
    ColTree: the tree information at the column
    RowData: the data item at the row
    RowTree: the tree information at the row
  Meta-template: meta_type

  Function:
    Merge(): combine two items in a FSL into one
*/
Begin
  For m=1 to n                /* n is the total number of FSL */
    /* construct meta-template for each column level */
    For ColLvl=1 to MaxColLvl
    /* MaxColLvl is the maximum column-level of the m^th FSL
      For ColSeq=1 to MaxColSeq
      /* MaxColSeq is the maximum sequence of the column level */
```

```
      Read ColItem   /* read a column data item from the mth FSL */
      Col_meta-template[m, ColLvl, ColSeq].Data = ColItem.Data;
      Col_meta-template[m, ColLvl, ColSeq].Tree = ColItem.Tree;
    Next ColSeq;
  Next ColLvl;
  /* construct meta-template for each row level */
  For RowLvl=1 to MaxRowLvl
  /* MaxRowLvl is the maximum row-level of the mth FSL */
    For RowSeq=1 to MaxRowSeq
    /* MaxRowSeq is the maximum sequence of the row level */
      Read RowItem   /* read a row data item from the mth FSL */
      Row_meat-template[m, RowLvl, RowSeq].data = RowItem.Data;
      Row_meta-template[m, RowLvl, RowSeq].Tree = RowItem.Tree;
    Next RowSeq;
  Next RowLvl;
  /* merge the Col_meta-template if there is more than one column level and its
    child and parent belong to the same tree */
  If MaxColLvl >=2
    For ColLvl= MaxColLvl to 2 step -1
      For ColSeq=1 to MaxColSeq
        If Col_meta-template[m, ColLvl, ColSeq].Tree =
          Col_meta-template[m, ColLvl-1, ColSeq].Tree
          Merge(Col_meta-template[m, ColLvl, ColSeq],
                Col_meta-template[m, ColLvl-1, ColSeq]);
        Endif;
      Next ColSeq;
    Next ColLvl;
  Endif;
  /* merge the Col_meta-template if they belong to the same tree
    but are in different sequences */
  ColLvl=1;
  For ColSeq=1 to MaxColSeq-1
    If Col_meta-template[m, ColLvl, ColSeq].Tree =
      Col_meta-template[m, ColLvl, ColSeq+1].Tree
      Merge(Col_meta-template[m, ColLvl, ColSeq],
            Col_meta-template[m, ColLvl, ColSeq+1]);
```

```
        Endif;
      Next ColSeq;
      /* merge the Row_meta-template if there is more than one row level whose
         child and parent belong to the same tree */
      If MaxRowLvl >=2
         For RowLvl= MaxRowLvl to 2 step -1
            For RowSeq=1 to MaxRowSeq
               If Row_meta-template[m, RowLvl, RowSeq].Tree =
                  Row_meta-template[m, RowLvl-1, RowSeq].Tree
                  Merge(Row_meta-template[m, RowLvl, RowSeq],
                        Row_meta-template[m, RowLvl-1, RowSeq]);
               Endif;
            Next RowSeq;
         Next RowLvl;
      Endif;
      /* merge the Row_meta-template if they belong to the same tree
         but are in different sequences */
      RowLvl=1;
      For RowSeq=1 to MaxRowSeq-1
         If Row_meta-template[m, RowLvl, RowSeq].Tree =
            Row_meta-template[m, RowLvl, RowSeq+1].Tree
            Merge(Row_meta-template[m, RowLvl, RowSeq],
                  Row_meta-template[m, RowLvl, RowSeq+1]);
         Endif;
      Next RowSeq;
   Next m;
   /* compare all Col_meta-template/Row_meta-templaete and eliminate the
      redundant ones */
   For m=1 to n
      Meta-template[m].ColData = Col_meta-template[m,1,1].Data;
      Meta-template[m].ColTree = Col_meta-template[m,1,1].Tree;
      Meta-template[m].RowData = Row_meta-template[m,1,1].Data;
      Meta-template[m].RowTree = Row_meta-template[m,1,1].Tree
   Next m;
   For i=1 to n-1
      For j=i+1 to n
         If Meta-template[i] = Meta-template[j]
            Delete(Meta-template[j];
         Endif;
      Next j;
   Next i;
End;
```

References

[1] C. Batini, B. Demo, A. Di Leva, A methodology for conceptual design of office data bases, Information Systems 9 (3/4) (1984) 251–263.

[2] T. Catarci, M.F. Costabile, S. Levialdi, C. Batini, Visual query systems for databases: a survey, Journal of Visual Languages and Computing 8 (1997) 215–260.

[3] J. Choobineh, S.S. Venkatraman, A methodology and tools for derivation of functional dependencies from business form, Information Systems 17 (3) (1992) 269–282.

[4] J. Choobineh, M.V. Mannino, J.F. Nunamaker, B.R. Konsynski, An expert database design system based on analysis of forms, IEEE Transactions on Software Engineering 14 (2) (February 1988) 242–253.

[5] J. Choobineh, M.V. Mannino, V.P. Tseng, A form-based approach for database analysis and design, Communications of the ACM 35 (2) (February 1992) 108–120.

[6] C.W. Holsapple, A.B. Whinston, Decision Support Systems: A Knowledge-Based Approach, West Publishing, St. Paul, MN, 1996.

[7] T. Isakowitz, S. Shocken, H.C. Lucas Jr., Toward a logical/physical theory of spreadsheet modeling, ACM Transactions on Information Systems 13 (1) (January 1995) 1–37.

[8] N.C. Shu, FORMAL: a form-oriented, visual-directed application development system, Computer (1985) 38–49.

[9] N.C. Shu, V.Y. Lum, F.C. Tung, C.L. Chang, Specification of forms processing and business procedures for office automation, IEEE Transactions on Software Engineering SE-8 (5) (September 1982) 499–512.

[10] D. Tsichritzis, Form management, Communications of the ACM 25 (5) (July 1982) 453–478.

[11] J.H. Wu, SDSS basis and application—a case study of the Taiwan Provincial Government, Journal of Chinese Institute of Industrial Engineering 13 (3) (1996) 203–213.

[12] J.H. Wu, A visual approach to end user form management, Journal of Computer Information Systems 41 (1) (Fall 2000) 31–39.

[13] S.B. Yao, A.R. Hevner, Z. Shi, D. Luo, FORMANAGER: an office forms management system, ACM Transactions on Office Information Systems 2 (3) (July 1984) 235–262.

Jen-Her Wu is Professor of Information Management at the National Sun Yat-Sen University. Prior to his doctoral study at the University of Kentucky, he received a BS degree in Industrial Design, earned an MS degree in Computer Science and worked as an engineer and manager in the manufacturing industry. Professor Wu teaches a variety of information management courses. His current research interests are in the areas of information systems development and management, human computer interaction and knowledge management. His research articles span such diverse journals as Information and Management, Expert Systems, Knowledge Acquisition, Decision Support Systems, Simulation Digest, International Journal of Expert Systems: Research and Applications, International Journal of Intelligent Systems in Accounting, Finance and Management, Journal of Computer Information Systems and others.

Her-Sen Doong is an Instructor of Information Management at the National Sun Yat-Sen University in Taiwan, where he received his PhD degree in Information Management. His research interests include electronic commerce and Internet communications. His papers have been published in the International Journal of Electronic Commerce and several conference proceedings.

Ching-Chang Lee is a lecturer of Information Management at Shu-Te University, Taiwan. He received his doctoral degree of information management from the National Sun Yat-Sen University (Taiwan, ROC). His primary research interests include decision support systems, electronic commerce and information systems development.

Tse-Chih Hsia is an instructor of Business Management at the Mei-Ho Institute of Technology. In the meantime, he is studying for his PhD degree in the Department of Information Management at National Sun Yat-Sen University. His areas of research include the information systems development and database management.

Ting-Peng Liang is professor of Information Systems at the Chinese University of Hong Kong and the National Sun Yat-Sen University in Taiwan. Prior to his current position, he had been the Dean of Academic Affairs and the Dean of the College of Management, Director of the Graduate Institute of Information Management and Director of the Software Incubator of the National Sun Yat-Sen University. He was also the founder of the Pacific Asia Conference on Information Systems. He received his doctoral degree in Information Systems from the Wharton School of the University of Pennsylvania and had taught at the University of Illinois and Purdue University until 1992. His primary research interests include electronic commerce, decision support systems and strategic applications of information systems. His papers have appeared in a number of journals, such as Management Science, MIS Quarterly, Decision Support Systems and Decision Sciences. He also serves on the editorial boards of several academic journals.