The THOR template editor

by KWANG-YA FANG

University of Illinois at Chicago
Chicago, Illinois

ABSTRACT

A form-entry system allows the user to view a database as a collection of data forms and therefore provides a user-friendly interface. Users can easily understand the information contained in forms and easily design forms of their own. The Template Handling On-line Reformatter (THOR) is an interactive software tool to support a form-entry system. The THOR system, designed for use at a terminal, provides a full-screen visual editor that allows the user to design different forms known as menus. To integrate THOR with a database system, the user may perform data-insertion, deletion, update, and retrieval through menu operations. This paper presents the general design of a prototype THOR system.
INTRODUCTION

The broad use of computers in different environments and the high cost of human resources in system development, maintenance, and application in recent years have redefined the definition of productivity. How fast a programmer can program is no longer a major concern of the data processing professional. Their attention is now focused on the human factors of computer systems with considerations such as: Is it easy to learn and understand? How little must the user do? How easy is it to do?

Over the last 10 years, the psychological profile of the computer user has drawn the attention of many researchers in industry and academia. Many have begun to study database query languages from the end-user's point of view. The motivation for such work has been diverse, but commonly reflects an attempt to define the notion of ease of use with some quantitative measurements. A survey of some of the experimental methods for evaluating query languages, and an elementary guide to how to read these kinds of experiments was recently presented by Reisner. As Reisner points out, there is a general tendency to overstate conclusions, but it is obvious that a natural representation of data at the user's level is a major factor contributing to the ease of use of a database system. This paper briefly describes a form-entry system which allows the user to perform database operations via form manipulations.

Form is a common human communication media. Users can easily understand the information contained in forms and easily design forms of their own. A form-entry system allows the user to view a database as a collection of data forms. A data form from the user's point of view is merely a related set of data fields displayed in a desired format. The Template Handling On-line Reformatter (THOR) is an interactive software tool to support a form-entry system. The THOR system, designed for use at a terminal, provides a special screen editor that allows the user to perform database operations by manipulating forms known as menus. This database interface provides the user with a "natural" view of data which means that the user is no longer required to use conventional coding.

The THOR system consists of three software modules, the Interactive Template Entry Manager (ITEM), the Interactive Menu Editor (IME), and the Interactive Data Entry Administrator (IDEA). The general design of the THOR system is shown in Figure 1. ITEM defines the Template Definition File (TDF). TDF contains data fields to be included in a template and the information relating a template to the underlying database. IME enables the user to perform full-screen editing to generate the Menu Specification File (MSF) from a TDF or an existing MSF. The Menu Specification File contains visual and "navigation" information to display a logical template in a user-desired format. IDEA supports data entry and retrieval. It provides the user with a set of easy to use menu manipulation commands to perform data entry, modification, and deletion. A subsystem under IDEA, the Database Interface Subsystem (DBIS), supports database access operations. A prototype THOR system has been implemented in C language on a VAX machine running under the UNIX operating system. The underlying database is currently a very primitive temporary system to demonstrate the operations of the THOR system. The general design of the THOR system is given in this paper with a brief description of the THOR interactive commands. Examples are given at the end of the paper to demonstrate the use of the THOR system.

FIELD TYPES AND DEFINITIONS

A logical template is defined as a set of data fields with the necessary information to link each data field to the underlying database. A logical template may be associated with one or multiple data files. In general, a logical template consists of data fields to be filled in, and field names that indicate what information should be filled in the data fields. There are three types of fields currently supported by THOR:

1. Primary field. A single piece of information to be entered (i.e., name).
2. **Structure.** A composition of several different primary fields (i.e., the structure date may contain three primary fields: month, day, and year).

3. **List.** A set of identical rows (elements). Each element has one or more primary fields and/or structures (i.e., the list student roster would have the fields: name, social security number, major, entering date, and grade point average. The field date is a structure, and the others are primary fields).

Each field in a logical template consists of two parts, field name and field data. Field numbers are automatically assigned internally by the THOR system. To help the user access individual fields during a menu design session, the field numbers may be displayed on the screen adjacent to their associated fields at the user's option.

A menu is a logical template with visual and "navigation" information so that it can be displayed in a proper format. In other words, a menu is a logical template displayed in a user desired format. Each menu can be displayed on one or more screen images. Each screen image consists of data fields with associated name and text strings, plus some reserved spaces for terminal operator communications (help and error messages, etc.).

### INTERACTIVE TEMPLATE ENTRY MANAGER (ITEM)

A logical template is defined by user commands in response to a series of prompts from the Interactive Template Entry Manager (ITEM). A Template Definition File (TDF) is created by ITEM to specify a logical template after the user has completed a template definition session. ITEM allows the user to create a new or modify an existing template definition file. When modifying an existing TDF, the updated logical template may be saved in a new file, or substituted for the original logical template in an existing TDF. Information collected by ITEM through these prompts include:

1. **Template contents.** Defines what data fields are included in a logical template.
2. **Data source.** Identifies the data files associated to the logical template.
3. **Data length.** Specifies the maximum length of each data field. If field length is not specified, the default value is the field length of the associated data field in the underlying database.
4. **Data format.** Selects one of two formats. Data fields in a logical template can be displayed in single- or multiple-line formats.

After collecting this information, ITEM performs certain database verifications against the underlying database directory so that queries to enter or retrieve data from the underlying database may be composed later by IDEA. For example, a key-field verification will ensure that all linked data files can be accessed.

In essence, a logical template is defined by ITEM with every field linked to the underlying database. A logical template, which may be further defined as a collection of data fields from one data record, a subset of one data record, or a conjunction of multiple data records or subrecords from different files, is independent of the logical or physical data layout of the underlying database. This allows the user to view the database as a collection of menus, and to perform all database interactions through menu operations.

### INTERACTIVE MENU EDITOR (IME)

The Interactive Menu Editor (IME) is a full-screen visual editor designed specifically for menu editing. IME starts the menu editing session with an automatically generated default display by reading a Template Definition File (TDF) or a Menu Specification File (MSF). In IME-mode, the user is allowed to visually define or modify menus. Because a menu is a defined logical template displayed in a desired format, no new fields may be added in this mode. Therefore, only visual and navigation information are to be added in an MSF.

One screen image of a menu occupies the top 19 lines. The bottom five lines of the display screen are used by IME for command entry and message display. Commands are entered by simultaneously hitting a control key and a character key. If there is any associated argument which must be entered to complete the request, an argument requesting message is shown on the screen. When a command is executed, the screen display is immediately updated, and the command entry area is cleared for new commands. The immediate feedback of a command’s effect on the display greatly reduces the effort required to design a menu, and makes editing easier to learn.

The menu editing commands are field-independent or single-field type commands. Field-independent commands are useful for setting global features in designing a menu. Normal full-screen operation commands are supported to allow the user to move screen images (page) forward and back, divide menus into pages, etc. Assistance commands display information that helps the user locate the position of each individual field (such as row and column coordinates), field numbers, and field positions. A field-selection command allows the user to select a target field for single-field operations.

The basic operation mode of the menu editor is field-at-a-time processing. Upon entry, IME starts an editing session in command-mode with all fields displayed. Normal editing operations are supported to allow the user to add, delete, modify, or replace characters and text. Single-field operations can be executed either by selecting a target field and performing the operation at the command line, or moving the cursor to the target position and performing the operation in place. A target field can be moved to any location on the menu by changing the coordinates of the field name.

### INTERACTIVE DATA ENTRY ADMINISTRATOR (IDEA)

The Interactive Data Entry Administrator (IDEA) provides database interface that allows the user to perform data entry
and retrieval at the menu level. IDEA operates at the user interface- and underlying database-interface levels. The design of IDEA at the user-interface level is very similar to that of IME and includes some special commands to support database operations. One basic difference between the two is that in data entry-mode, the user is restricted to manipulating field data, and is not allowed to change field names or move fields around; in menu editing-mode, the user cannot access field data.

Three different types of commands are available in IDEA, single-field, field-independent, and menu. Single-field and field-independent commands such as search data, substitute data, change data, or enter data provide data manipulation operations for individual fields. Also included are some commands specified in the menu editor which locate fields and move screen images. IDEA menu commands are for database operations such as inserting, deleting, updating, and reviewing menus.

Upon entry, IDEA displays a menu with no data. The bottom of the screen is reserved as the command-entry area. Data may then be loaded from the underlying database by a user specified menu key. This may be a single or a set of keys. If multiple keys are required, IDEA automatically requests that the user enter all key values. If a single key value is entered, an individual menu may be retrieved from the database. By specifying a range of key values, multiple menus may be retrieved for reviewing. The user may also perform data insertion, deletion, and updating through data entry commands in this mode.

The underlying database interface is performed by an independent subsystem, the Data Base Interface Subsystem (DBIS). Because DBIS is the only module which depends on the underlying database, THOR has the flexibility to be integrated into different database systems by rewriting or modifying the DBIS subsystem.

For most database management systems (DBMS), different levels of abstractions of the database have been used to describe its design and application. A logical template or a menu can be viewed as another abstract representation of a database. To integrate THOR with a DBMS, the DBIS must be able to transform menus into lower-level database abstractions. The main concern is to transform a given form into the database conceptual schema, and vice-versa. Query language can then be generated automatically for data entry and retrieval.

Currently, a simple DBIS has been implemented interfacing with a very primitive database system to support and demonstrate the THOR system. In this prototype system, DBIS directly maps logical templates into the physical database. Because a logical template may be a composition of multiple underlying data records or subrecords, the DBIS first decomposes a logical template into "masks" that correspond to the associated data records. After the decomposition, each mask can be mapped in a one-to-one fashion to the underlying data record. If a mask contains only some of the fields from a record (i.e., a subrecord), only those fields in the mask will be included in the mapping. Conversely, multiple data records may be mapped into masks and then merged together to form a logical template.

**EXAMPLES**

This section briefly demonstrates the use of the THOR system in interface with an underlying database. Some features and commands are not included.

Assume that a logical template has been defined in a TDF to track the status of a consulting company's software development project. Within the logical template, information relating to the project has been specified to include the following:

1. **Primary fields**—Project Number, Project Name, and Description.
2. **Structure fields**—Project Manager, Project Status, and Project Environment.
3. **List field**—Project Developers.

This logical template can be displayed in its default format when entering IME as shown in Figure 2. Note that each

```
1 Project number: __________
2 Project name: ____________
3 Description:

4 Project Manager:
  5 name: ____________
  6 dept no.: ________
  7 phone: __________

8 Project Status:
  9 project phase: __________

10 number of developers: ________
11 budget: __________
12 completion date: __________
13 review date: __________
14 comments:

15 Project Developers:
  16 developer1
    17 name __________
    18 title __________
    19 project phase __________
    20 hourly rate __________

  21 developer2
    22 name __________
    23 title __________
    24 project phase __________
    25 hourly rate __________

  26 Environment:
    27 Hardware Required: __________
    28 Software Required: __________
    29 Vertical Market: __________
    30 Comments: __________
```

Figure 2—Automatically generated default menu
structure field is composed of several primary fields, and that
the list field is a list of structures. For example, the structure
Project Status (field 8) consists of the six primary fields project
phase, number of developers, budget, completion date, review
date, and comments. The list field Project Developers (field
15) contains two structures for two different developers; each
such structure contains information fields for each developer
(name, title, project phase, and hourly rate). This logic tem­
plate includes the data files project file and employee file. The
project file contains all information related to a project in­
cluding project developers' names. The project developer's
name also serves as the key field to the employee file. There­
fore, information may be retrieved or entered by the project
number through a project status logical template or menu.

The default menu display may be reformatted to a user
desired format by moving fields around and changing field
names. For example, the Project Manager field may be modi­
fied so that the field name is displayed in the center with three
primary fields displayed on one line, as shown in Figure 3.
Similarly, the list field Project Developers has been re­
formatted so that four primary fields are displayed on one
line. In Figure 4, duplicated field names for each developer
are removed and the menu is displayed in a more user-desired
format. The field names in a menu may also be changed; they
do not have to be identical to the names defined in the logic
template. This gives the user the flexibility to select the field
names in the display, and does not limit the user to field names
as defined in the database.

After the user completes menu editing, a properly for­
matted menu may be saved in a Menu Specification File for
future data entry operations. The menu specified in Figure 4,
for example, can be saved with a menu name such as
project[ spec].

With a defined menu, the user may enter data entry-mode
by specifying the menu to be used. IDEA first displays the
menu without data on the screen (as in Figure 4). The user
may then perform data insertion, deletion, and update
through data entry commands. To retrieve data, the user must
first enter the value(s) for a key field(s) and then execute a
load data command. In this example, Project Number is the
menu key. Multiple menus may be retrieved by entering a
range of key values (i.e., 1000 < . AND. < 1050). The user may
insert data into the database by filling out the empty fields and
executing save data command. The user must enter data in the
key field(s), but is not required to enter data in every field in
a menu for insertion. Each empty field is treated as data
absent and is stored as an empty field in the database. A data
update is done by first retrieving data from the database,
changing or adding field data, and then executing a save data
command to update the underlying database. In Figure 5, the
project information for project number 1005 has been loaded
into menu "project.spec" and displayed. Note that all internal
field numbers are automatically removed when the final menu
is displayed.
The THOR form-entry system is a prototype developed in conjunction with a research project on the database operating environment. The principal objective is to design a database interface system in consideration of user behavior. Generally, users engage in goal-oriented activity; they always attempt to accomplish their goals as effortlessly as possible within the constraints imposed upon them.\(^5\)

THOR is implemented to demonstrate user operations in data entry and retrieval. With a prototype system in place, user interaction and database operation via form manipulation can be studied. Further work in the transformation of forms into conventional query languages to integrate THOR with existing database systems is needed.

Different approaches have been proposed to make use of forms as an integral means of software application.\(^7\)-\(^10\) Most are concerned with the development of integrated office information systems and do not provide a general query language facility. The work of Embley's Form Processing System (FPS) introduces the notion of strong interpretation to map forms into relational algebra queries, thereby making it possible to extract data from a relational database and display it on a form as expected. In Embley's study, the transformation is concentrated on relational database systems. Further studies are needed to investigate the transformation for other models of database systems. An interesting and challenging work associated with form transformation is the study of performance enhancement, including query generation and optimization.

The integration of other database features such as intra- and inter-form checking, and data verification should be studied. The goal is to develop a database operating environment emphasizing user behavior. The concept is to allow the user to interface with a database so that the user sees what a database is. As a result, minimal user knowledge about database and query language is required before the user can start using a database system. Because the user is performing database queries in a natural way of human thinking ("what you see is what you get"), it should give the user a high degree of confidence that his/her queries are correct. Furthermore, because in most cases there are no programs to write, it should be faster to perform queries in THOR system than in conventional query languages.

Many researchers believe that improvements in the relationship between computer systems and users will play a very important role in the development of computers. Difficulties in user interface have been the major obstacle to user acceptance of computer systems. The power of computers can be dramatically enhanced with a new type of software that provides a friendly operating environment and functions as an intermediary between computers and their users.

**ACKNOWLEDGMENTS**

The author would like to acknowledge W. Liu and W. P. Koo for their help in developing the prototype system.

**REFERENCES**
