Fagen : A *F*orm-based *A*pplication *Gen*erator

Rattakorn Poonsap[*] *and Wittaya Watcharawittayakul*[**]

1. Introduction

Presently, human factor has been considered important in designing computer systems. Specifically, man-machine interface is no longer the last job in developing software. Usage of computers has grown dramatically in laboratories, offices, and homes. Studies of user interface can be done for both hardware and software. Hardware engineers may explore keyboard design, graphic displays, pointing devices, and voice recognition, whereas software developers may be interested in screen design, menu selection techniques, natural language processing, and visual programming.

Interactive software can use various techniques to build user interface; e.g. multiple displayed windows and pull-down menus. All of them base on the model of human-computer interaction [Norm84]. In this model, one follows 4 stages to accomplish a task; namely, forming the goal or intention, reviewing possible actions, selecting the most appropriate one, executing the selected action, and evaluating the outcome.

Task-oriented user interface must help users get their jobs done without confusing them with programming language syntax. The study of human-computer interaction is inter-disciplinary and involves many areas; e.g. computer science, cognitive science, psychology, and ergonomics.

This paper presents a *F*orm-based *A*pplication *Gen*erating software called *Fagen*. In general, *Fagen* will generate menu and dialog driven application programs. For software development, one must construct 3 building blocks: the user interface, database, and data processing programs. Currently, database manipulation is supported by database management system (DBMS), whereas the user interface and data processing programs are usually implemented by programming languages. Although user interface and application program generators exist, they are primarily used for prototyping because the generated codes are inefficient. Furthermore, these tools are usually integrated as a single system which is difficult to link to others.

The development of *Fagen* has attempted to eliminate the gap by automating the user interface implementation. Consequently, the development life cycle can be shortened significantly. Furthermore, *Fagen* also provides data processing functions via its pre-defined database operations.

Presentations of this paper are as follows. Section two describes the motivation, goal, and methodology of the research. Section three briefly outlines the software requirement. *Fagen* architecture follows in section four, whereas section five explains the

[*] Instructor, School of Applied Statistics, National Institute of Development Administration

[**] Vice president, Stock Exchange of Thailand

idea of description languages for various forms. Section six illustrates how to user *Fagen* to develop applications. Finally, section seven presents the future work and conclusion.

2. Motivation, Goal, and Methodology

In 1992, the Thai computer market was worth more than 16 billion baht. Software shared about 25 percent of this amount. The share continually increases as hardware cost drops. Combining with telecommunication, the hardware market is the largest among electronic industry. All these equipment's require controls by software.

For computer hardware, the workstation market grows fastest, whereas that of personal computers follows. In the coming years, as price-performance ratio improves, the 2 markets will combine. All these computers need at least a few software. First an operating system is required. Then other software is installed depending on applications.

In Thailand, nearly all software packages are imported. This may not seem serious because of the lack of the software copy right law. However, toward globalization, the law will be passed. The loss of revenue due to these so called pirate software packages was estimated to be over 1 billion baht in 1990.

The price advantage of pirate software has a negative effect on local software industry. The growth is slow and in the long term the country can be far behind in software development technology. According to the Department of Commercial Registration, the number of software companies was close to 100 in 1991. This number is far less than those in other newly industrialized countries; e.g. 300 in Taiwan and 700 in South Korea.

Considering the very large software market, we need to have our own software products, both for internal use and export. Unlike other electronic sectors, the knowledge for software development is well-known. Only experience to deal with large software projects is needed to stay competitive in the market.

For software, two markets can be classified. First is the front-end market where applications operate. Apart from tailor-made software, these software packages includes office automation (word processor and spreadsheet), data communication, accounting system, computer aided design, statistical package, etc. The back-end market includes products like software development tools (compilers, DBMS, utilities), graphics libraries, and operating systems.

The direction for Thailand should be in the front-end market where customized products are needed. Experience from developing these products should bring the industry toward more generic applications and back-end products. Most customized and tailor-made software is for form-based data processing applications. To develop this kind of software, one needs to know software development steps and project management. Within the software development steps, defining and creating user interface is probably the most important factor to determine the success or failure of the products.

Currently, most form-based application development tools on microcomputers; e.g. Dbase and Foxbase, include components allowing designers to construct the user interface. However, most are limited and still need programming effort to create menus and dialogues in the program code. This method is not flexible and may cause difficulty in debugging. Although the mentioned products also provide automatic menu and dialog generators, they are integrated as a single system that cannot work across products.

This research is to develop a form-based application generating software so as to separate user interface part from the data processing part. The advantages to do this are:

1) Software developers can concentrate on separate designs for user interface and data processing.
2) The user interface program is independent from the application program.
3) For software prototyping, the user interface part can be created and agreed upon early in the project.
4) It is easy to modify, maintain, and manage.
5) To establish the standard for user interface development.

It is expected that this software will become beneficial to software developers as a tool to help develop the significant component of any software packages. The time saved from the development cycle and the improved quality of products should justify the study and investment of this research project.

The C programming language is chosen to implement the system. Part of it will be implemented as library so as to be portable and able to link to other programming languages. The system is developed using the exploratory approach. Initially a small workable system is developed. Improvement and more functionality are gradually included in the system as more experience is gained from review and usage. The exploratory approach prevents us from laying down the detail requirement and specification at the beginning when there are too many constraints, alternatives, and criteria. Furthermore, we consider the system not a typical application to which a conventional developing approach is applicable.

3. Software Requirement

Users must be able to define menus, dialogs, and tables as text files. These files will serve as data to the generator for generating C code. Description languages are designed for describing menus, dialogs, tables, and database. Furthermore, a front-end software will be built to permit designers to draw their menus, dialogs, and tables directly on screen and translate the drawings into the description languages.

For menus, three types can be defined. They are horizontal menus, vertical menus, and pull-down menus. These menus stay at specified locations on the screen. A menu shows pre-defined texts that represent choices of actions.

Beside menus, users can also define dialog boxes, look-up tables, and database interface. A dialog box serves as a form for data entry. Users should be able to define the format and data type of each field. A look-up table is a list of items that can be selected to fill in a field of a dialog box. This will minimize memorization for the end users. Database interface is a way to specify the relationship between the database and a dialog box and table. This will help control grouping of database components and functions. The database interface must be general enough to allow designers to select any data management software; e.g. DBMS or conventional file management.

The design will try to explore object oriented design style because preliminary study shows that the system can be viewed naturally as collection of objects; e.g. menus, tables, and dialog boxes.

4. System Architecture

Figure 1 shows the architecture of *Fagen*. The system was implemented as four layers. Each layer provides fundamental functions for the above layer; i.e., higher abstraction that is easy to use and maintain.

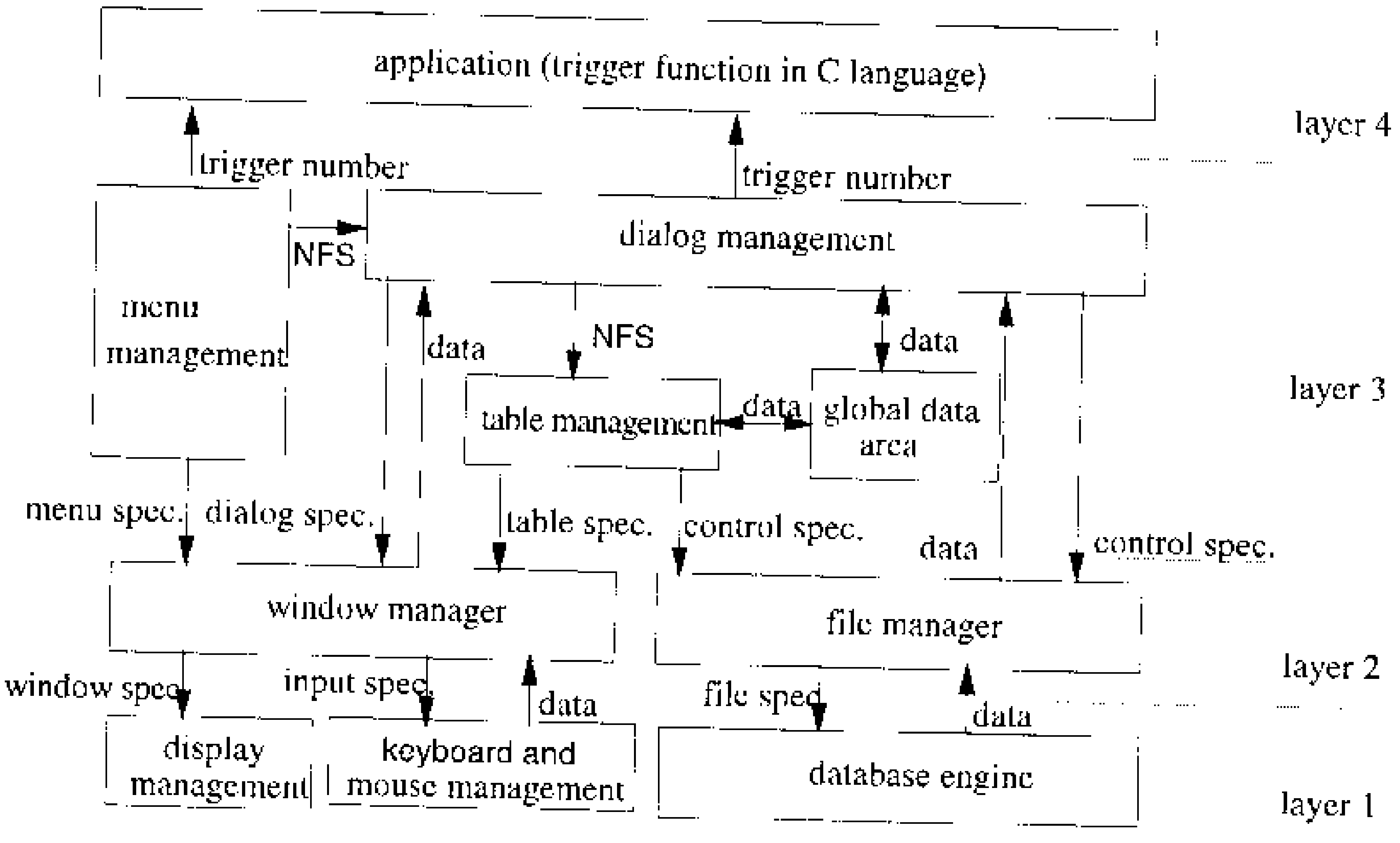

Figure 1. Architecture of the menu generator

Two components are at the lowest level. First is the basic I/O support that interacts with users via the display, key board, and mouse. Second is the database engine that supports data management. Above the basic I/O support is the window manager that supports multiple window management. Similarly, above the database engine is the file manager that should present a unified access method for any database engine. The third layer is the menu, dialog, and table management. All three components communicate with the window manager to set up parts of the screen as working areas called windows. The dialog and table management also interacts with the file manager to interrogate data from the database.

The highest layer consists of routines (C functions) that are application-specific. These routines are called trigger functions. From a menu or a dialog, application may need some actions for responding to some specific input. These actions will be implemented as C functions and each has a number, called a trigger number, associated with it. From the menu and dialog management, control can be passed to a trigger function by using its trigger number. Generally, trigger functions serve as a back door of the menu generator.

Generally, the menu management provides choices of functions for user's selection. The dialog management provides a two-way communication between application program and users. General usage of dialog is for data entry, and database viewing. Finally, the table management offers a mechanism to list data from the database. The purpose is to serve as a memorandum for data entry.

5. Description Languages

Four description languages are designed to describe menu, dialog, table, and database. Each menu, dialog, and table description is kept in a text file, whereas the database description is stored in many text files. These text files can be created and modified by any text editor. For an application, these files must exist at run time because *Fagen* will read them for building its internal data structures. The approach may be inefficient but is flexible for modifying description files without recompiling the whole application.

1. *MDF* = *frame* + {*item*}
2. *frame* = *title* + *no_of_item* + *menu_type* + *top_corner* + *bottom_corner* + *border_type* + *title_style* + *title_position* + *title_attr* + *menu_before_func* + *shadow* + *border_attr* + *background* + *text_attr* + *sel_char_attr* + *non_select_item_attr* + *bar_attr*
3. *title* = character string /* menu title */
4. *no_of_item* = 1..23 /* number of selectable items */

•

•

•

20. *item* = *item_position* + *item_name* + *quick_sel_char* + *unique_tagid* + *feature_mask* + *item_sel_func* + *before_item_func* + *after_item_func* + *help_num* + *action* + ([*file_name* | *command*])

•

•

•

Figure 2. Definition of Menu Description Language

Since the syntax and meaning of each description language is lengthy, only some parts of the menu description language is presented. The rest is similar and can be found in [PoWa93]. Figure 2 shows the definition of the menu description language. The notation explaining the language follows that of [Your89] for defining data dictionary. The symbols used and their meaning are:

Non-terminal identifiers; i.e. those that must be defined further, are shown in italic face, whereas terminal identifiers are shown in normal face. Note that the description language is a list of attributes the positions of which determine their meaning.

The description is listed in components, each of which is numbered accordingly. An MDF (Menu Description File) consists of a *frame* (component 1 to 19) and many *items* (from component 20). *Frame* describes the appearance of the window opening for the menu, whereas *items* correspond to choices within the menu.

Symbol	Meaning
=	is composed of
+	and
( )	optional
{ }	iteration
[ ]	select one of several alternatives
/*...*/	comment
|	separates alternatives in the [ ] construct.

A *frame* containing the menu is bounded by a rectangle whose top-left and bottom-right corners are described by *top corner* and *bottom_corner* respectively. These positions are the row and column positions in the text mode. *Border_type* signifies various border line styles; e.g. single or double line, of any color (*border_attr*). The window may appear to have depth by the *shadow* attribute. *Background* specifies the color of the window background area. *Title* is a text string serving as the header of the menu. *Title_attr* specified its color. Its position is specified by *title_position* which locates it at either the left top, center top, or right top of the menu. *Title_style* allows it to be at or below the top border line.

No_of_item specifies the number of choices in the menu. Hence, *item* must iterate this number of times. Description from component 20 is used to characterize an *item*. This must be repeated as many as the number of *items* in the menu. General *item* attributes include its position (*item_position*), text description (*item_name*), the quick selection character (*quick_sel_char*), help message upon request (*help_num*), and before and after selection actions (*before_item_func* and *after_item func*). When an item is selected, *action* specifies what the application will do. It can be no action, calling a C function (*item_sel_func* as trigger number), executing another program, making a system call, or activate another form.

Description languages for dialog, look-up table and database are similar to that for menu. Specifically, the dialog description language is the most complex one since it needs interactions to the database. One uses dialogs for data entry and information retrieval. A dialog consists of one or more related blocks. The concept of related blocks results directly from many data processing applications where accessing one file requires accessing more detail from other files; e.g. retrieving a purchase order record form the purchase order master file needs retrieving detail item records related to the purchase order from the detail order file. In the example, the purchase order record may be displayed in one block, whereas its detail is displayed in the other. Changing the order record identifier (key field) in the former block should automatically reflect the detail change in the latter block.

A block may tie to one or more files in the database. All the files associated with a dialog constitute a File System File (FSF). Hence, each dialog is served by one FSF. In a FSF, one is called the block master file. A block master file is the file that can be both read and written. Other files can only be read.

A block consists of fields and static texts. A field is a place where data must be filled in either from the user, file, system, or combination of other fields. Static texts are just description that appears as prompts to the user.

6. How to Use *Fagen*

Figure 3 shows the *Fagen* environment from the end-user perspective. From this perspective, users view *Fagen* as 2 components: the menu tree editor and the application generator. The menu tree editor is the front-end that interacts with the users. It allows users to draw their applications graphically. Specifically, an application is a tree a node of which is a menu or a dialog. Links in the tree represent its control flow structure.

An application can be incrementally built using the menu tree editor. Partial work will be saved as graphical representation and description files. When the work is done, all description files will be used by the application generator to create the executable program. Data processing and database accesses can be specified when a node is created; i.e. associated with a menu item or a dialog field. These operations can be implemented by using *Fagen* built-in functions or as trigger functions in C.

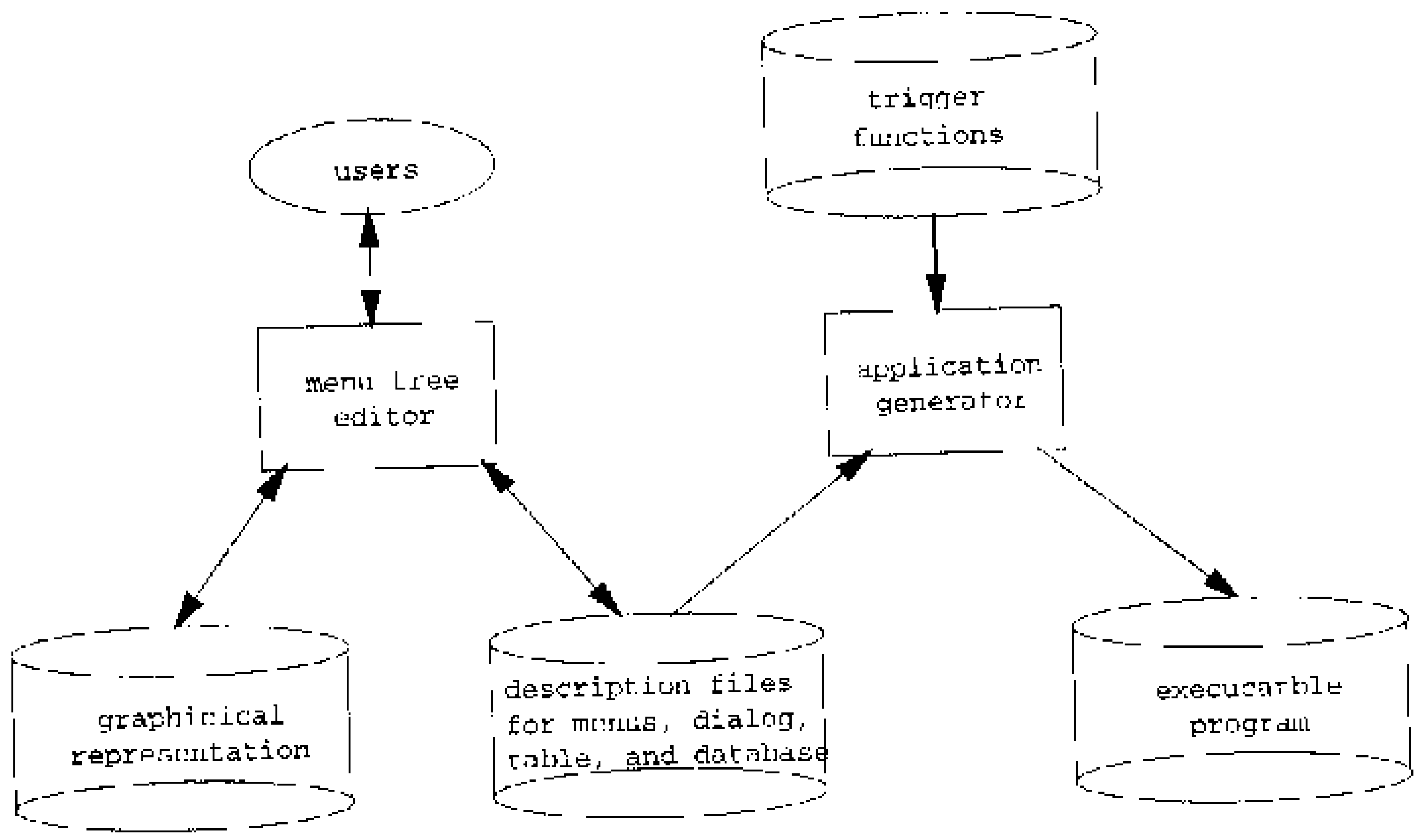

Figure 3. *Fagen* from the End-user Point of View

The menu editor is a window-based environment that is easy to use. Graphical user interface techniques are exploited to transform the application domain to the complex syntax of description languages. Experienced users who require more flexibility can bypass the menu tree editor and create the description languages directly by using any text editor.

Generally after the requirement and specification phase, designers commence designing the software structure. For form-based applications, the developing steps are as follow:

1) Designing the database structure.
2) Designing the menu tree.

3) Designing the detail of each menu and dialog.
4) Coding program.

Step 1, 2 and 3 can be done by using the menu tree editor, whereas step 4 is automatically done by the application generator. An example of details of these steps can be found in [PoWa93].

7. Conclusion and Future Work

In this paper, a form-based application generating software is presented. The function of the software is to help automate development of menu and dialog driven applications. In the proposed environment, one will be able to create applications by first building its menu tree. This tree serves as a control-guidance in the application. A menu tree consists of nodes and links. A node represents a menu or a dialog, whereas a link specifies a control path. Both menus and dialogs become data to the application program instead of embedded code in it. Consequently, the user-interface part and the data-processing part are separated in an application. Hence, a user-interface prototype can be rapidly created and evaluated. Beside user-interface prototyping, the system also supports data processing via trigger functions and database accesses. For an end-user action; e.g., menu selection or specific detected key press, the system can perform an associated pre-defined C function or database operation. Specifically, a database operation can be linked to each field in a dialog.

Presently, the language description recognizers for menu, dialog, table, and database and the application generator are complete. The interface to a database engine[1] is also complete. With all these components, form-based applications can be developed.

The remaining work is to develop the front-end that permits users to graphically create the menu tree and fill in details of each menu, dialog, and table. This front-end should transform graphical representation to the description languages. The Microsoft Windows environment has been chosen to implement the front-end. Future work should also include designing a uniform or standard interface to any database engine and developing a report generator.

Finally in the perspective, we have observed the market situation of similar software. Apart from those high-ended integrated database systems, only a few data management software on the personal computer platform are available with menu generating capability. The trend toward this direction is undoubtedly obvious. Automatic graphical user-interface is a necessity at the front, whereas client-server with standard data management interface is mandatory at the back.

REFERENCES

D.A. Norman. 1984. *Stages and Levels in Human-machine Interaction.* **Intl. Journal of Man-Machine Studies** 21 : 365-375.

R. Poonsap and W. Watcharawittayakul. 1993. **A Software for Menu Generator.** Progress Report submitted to the Research Promotion Committee, National Institute of Development Administration.

E. Yourdon,. 1989. **Modern Structured Analysis.** Prentice-Hall, Inc.

[1]In the project, the Paradox engine was selected.