A DATA-DRIVEN USER INTERFACE GENERATOR FOR A GENERALIZED MULTIPLE CRITERIA DECISION SUPPORT SYSTEM

**Matthias Jarke, Mohamed Tawfik Jelassi
and Edward A. Stohr**

October 1984

Center for Research on Information Systems
Computer Applications and Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #82

GBA #84-72(CR)

A shorter version of this paper appears in **Proceedings IEEE Workshop on Languages for Automation**, New Orleans, November 1-3, 1984.

1.0 INTRODUCTION

The development of reliable user friendly software is a difficult and error prone task. And yet, as amply demonstrated by the current wave of popularity of spread-sheet programs, the rewards can be great. Software that combines functionality with a good interface will be used by management personnel on a voluntary, everyday basis and in ways that, pernaps, were not even dreamed of by the software designers. The applications that have so far been developed have, for the most part, been fairly rudimentary from a management science point-of-view. However, we believe that the existence and popularity of these elementary models will inexorably create a demand for more sophisticated ones. In fact the processing of information in more and more complex ways will become a major focus of economic competition and survival.

In this paper we describe a software system that is designed specifically to facilitate the development of decision support systems (DSS). Our objective is to help the management scientist build successful software. Our focus is on DSS employing Multi-Criteria Decision Making (MCDM) models. But this is not the only possible area of application. Since our goal is to develop software that will help others develop software there is a possibility for confusion. To clarify the discussion we will call the set of software tools that we are designing a "Generator" for multi-criterion decision support systems (GMCDSS). The generator provides an environment for model builders to develop "target" software in the area of multi-criterion DSS (MCDSS). The builders therefore are the users of the GMCDSS while the decision makers use the MCDSS.

The GMCDSS provides a set of languages and a uniform environment for the model builders. It helps them interface many different kinds of MCDM algorithm and provides screen generation and database facilities for direct employment in the target MCDSS. It is based on an extension of normal database management techniques in which (1) application data, (2) management science models (and other programs) and (3) meta-data concerning the structure of decision-making problems, are combined in a uniform formalism. Our approach is to model the decision processes of the end-users using a data abstraction hierarchy. It is at this point that the generator becomes specialized to a particular class of application (in our case MCDM problems). Once the data abstraction has been correctly defined it is much easier to build a target system which will provide a uniform and friendly environment for the user. Essentially we are following the ROMC (Representation, Operations, Methods and Control) approach to DSS building first advocated by Sprague and Carlson [1982]. In our case the "representation" involves principles of data abstraction.

Section 2 of the paper briefly describes the general environment of MCDM decision-making and develops a set of software design requirements. Section 3 outlines the architecture of the data manager component of the GMCDSS and its relationship to the overall system architecture which was described in a previous paper [Jelassi et al., 1984]. Section 4 describes the abstraction hierarchy together with some extensions of the normal data definition component of relational DBMS's. Given this data model Section 5 describes the process of developing the user interface by means of an example. This description is accompanied by samples of the code used by the model builders. Essentially this employs the SQL language [Astrahan et al. 1976] with two different kinds of extensions: (1) to allow us to take advantage of the meta data stored in the abstraction hierarchy, and (2) to allow us to employ database techniques to dynamically generate user screens representing the current state of the man-machine decision system. Finally some conclusions and suggestions for future work are provided in Section 6.

2.0 DESIGN REQUIREMENTS FOR MCDM MODELS

There are a wide range of different MCDM techniques. All are designed to help cope with the existence of multiple, conflicting objectives in decision problems. Some of these methods (for example, goal programming [Ignizio 1976] and UTA [Jacquet-LaGreze and Siskos 1982]) deal with quantifiable objectives and constraints and represent extensions of single criterion optimization techniques. An important group of techniques is based on utility theory [Keeney & Raiffa, 1976]. Other techniques such as AHP [Saaty, 1980] and ELECTRE [Roy, 1968] are designed to handle situations where judgements have to be made between alternatives on the basis primarily of qualitative, relatively uncertain, information. Correspondingly, the output from the system ranges from a complete specification of the levels of a number of activities (goal programming), to a cardinal priority ordering (AHP) or simply a dominance ranking (ELECTRE).

Good reviews of MCDM techniques are contained in [Zeleny 1982] and [Bui, 1984]. For our purpose it is sufficient to observe the following:

1. All of the techniques rely on subjective inputs from users;

2. They attempt to provide informational and computational support but do not dictate the final decision;

3. Often the methods involve extensive interactions with users in which learning takes place and either utility functions are derived or "dominated" alternatives are successively eliminated;

4. There are many different techniques and many different situations in which they might be employed;

5. Any given decision situation might require that several techniques be used in conjunction;

6. End-users will usually be inexperienced both in MCDM techniques and in the use of the computer.

As an example of (5), it is conceivable that a group decision-making problem may start-out as a cooperative one but gradually become uncooperative. Since the reverse is also true the need for a versatile, intelligent system becomes obvious. It is also apparent that the system should be able to play an "advisory" role in helping potential users choose suitable techniques to apply to their decision problem. Finally, the delivered MCDSS should also have a helpful and easy to use interface and exhibit other properties of good software such as accuracy, reliability and maintainability.

An architecture for a system that will help provide these capabilities is described in Section 4 below. Before leaving this section, however, we note that a typical use of a MCDSS by an end-user will involve the following phases:

(1) Selection of an existing application (for which data and a history of prior usage exists) or alternatively the definition of a new application area.

(2) Selection of one or more MCDM techniques from a "model bank" of algorithms maintained by the GMCDSS.

(3) Gathering of data (perhaps from external sources) concerning the application area.

(4) Selection and/or computation of criteria by which the relative merits of the alternatives are to be judged.

(5) Restriction of the set of possible alternatives to be considered on a priori grounds.

(6) Generation of data consistent with steps (4) and (5) in a format acceptable by the models selected in step (2).

(7) Interaction with the model in the solution of the MCDM problem.

(8) Storage and analysis of intermediate and/or final results.

The data abstraction hierarchy on which the GMCDSS is based explicitly recognizes the above decision-making steps. This model is described in Section 5.

3.0 ARCHITECTURE FOR DATA MANAGER IN GMCDSS

Figure 1 shows the components of the Data Manager sub-system of the GMCDSS with which we are concerned in this paper. As discussed more fully in [Jelassi et al., 1984], the Data Manager connects to two other major subsystems--the Model Manager and the Dialogue Manager. At this level the design resembles that proposed by a number of other researchers (e.g., [Sprague & Carlson, 1982], [Stohr & White, 1982]).

Briefly the Dialogue Manager provides menu management, screen generation and graphics facilities. It also contains information on physical device characteristics such as line speeds, screen sizes and communication protocols. It provides two-way communication with both the Model Manager and the Data Manager. The data flows between the Dialogue Manager and Model Manager consist of prompts from the MCDM models, and reciprocating commands and parameter values from the end-users. The data flows between the Dialogue Manager and the Data Manager consist of queries and update transactions made by the user and reciprocating query answers, confirmation messages and data dictionary definitions from the Data Manager.

The Model Manager consists of executable modules (MCDM models and general service programs) together with modelling language facilities and execution management. The data flows between Model and Data Manager consist of dynamic requests from the models for information and the corresponding responses from the DBMS. In addition, as explained in this paper the Data Manager supports the user in generating the data (goal programming tableaux, matrices of criteria values etc.) required by the MCDM models.

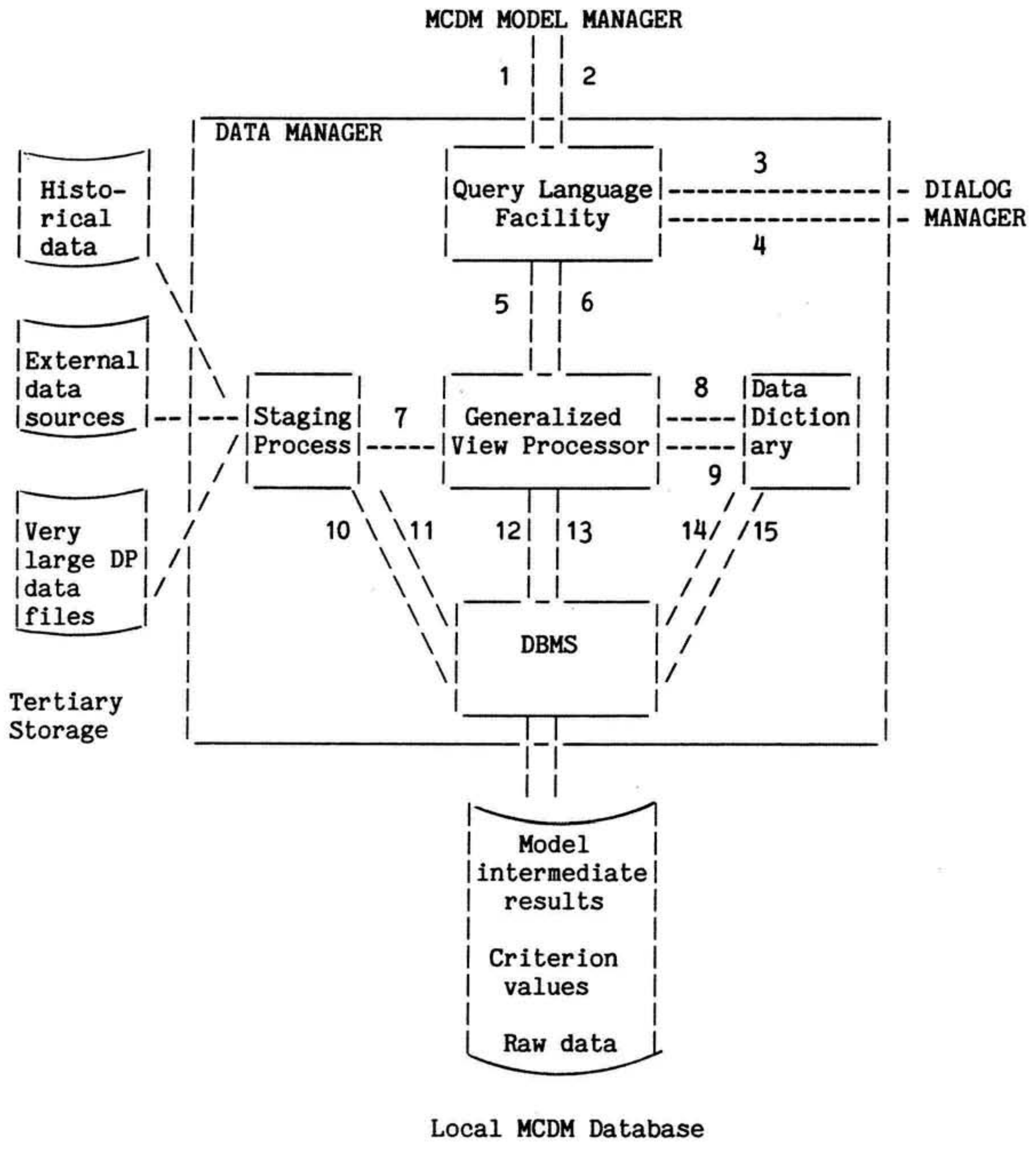
The Data Manager consists of the five software components shown in Figure 1. The Query Language Facility is the main focus of this paper. This is the language translator and message switching center of the GMCDSS. It links the three major sub-systems and allows them to be developed somewhat independently.

The internal details of the Generalized View Processor will be described in more detail in another paper. However, its function within the total GMCDSS will play an important role here. In relational database theory a "view" is generally a virtual relation that is defined by a query addressed to the "base relations" of the database. The view definition is stored but not the resulting database table. Users may compose queries in terms of either the views or the base relations of the database. A view facility provides a number of advantages. First, expressing queries in terms of views often increases the expressive power of the language by providing a kind of short-hand notation. Secondly, certain users may "know" the database only in terms of the views that are relevant to their needs. This simplifies the user's learning task and can be used to provide an important measure of security. As illustrated later in the paper, we find a need to generalize the accepted concept of a view to allow views to be parameterized and to include not only raw data but also computed values.

The Data Dictionary will provide the system with "self-knowledge". This facility must also have enhanced capabilities since we will require it to handle metadata. Its role (but not its implementation) will be discussed below.

The Staging Processor provides facilities for loading external data from heterogenous sources into the DBMS. In a microcomputer environment this might be a decoupled "file-server" allowing communication with the company's mainframe and/or external information utilities. We will not consider this component further in this paper.

The final component of the Data Manager is a general purpose relational data base management system (DBMS). We will assume that this will support interactive querying as well as embedded query languages. To provide a concrete example we will assume that the DBMS supports the SQL query language.



Legend:

1. Query/Insert
2. Data/Definitions
3. Query/Insert/Delete/Update
4. Data/Messages/Definitions
5. Translated transaction
6. Preprocessed data
7. Request
8. Request
9. Criteria definitions/View definitions
10. Load
11. Unload
12. Database transaction
13. Raw data
14. Request
15. Data definitions/Integrity constraints

Fig. 1: The Data Management Component of the GMCDSS

4.0 ABSTRACTION MECHANISMS IN MODEL/DATA MANAGEMENT

Abstraction mechanisms have been widely proposed for data modelling since the original papers by Smith and Smith [1977]. In this paper, an abstraction mechanism will be introduced that allows the system to guide the user in a stepwise refinement process from the selection of an application and a decision model, to the choice of a subgroup (category) of alternatives to be considered, to the choice of evaluation criteria for that category, and finally to the extraction of decision-relevant data from an underlying database, followed by the execution of the model.

This process is implemented as a sequence of menus. However, in contrast to typical menus, their format is not rigid, but depends on data retrieved from the database at various abstraction levels. In this section, the abstraction hierarchy and its representation in a slightly extended relational model will be presented. In Section 5, its use for data-driven user interface generation will be demonstrated.

4.1 Abstraction Hierarchy For Model Selection And Data Extraction

Figure 2 depicts the abstraction hierarchy that plays a central role in the GMCDSS. For simplicity, we assume that the MCDM method requires as input a set of alternatives each characterized by a number of properties or attributes. For example, in a car-buying example the alternatives are types of cars and the base relation would contain relevant attributes such as "maximum speed", "fuel consumption" etc. Some (but not necessarily all) of these attributes may be important to a particular user's decision problem. The attributes in this subset are called "criteria". The data for the decision problem is stored in a "decision matrix" where the rows represent alternatives and the columns criteria.

The main assumption of the abstraction mechanism is that all necessary information about alternatives is derivable from data stored in an underlying database. (However, during the execution of a particular model the user may be allowed to add additional manually defined criteria and alternatives.)

The abstraction hierarchy uses several types of abstraction: aggregation for combining the input from multiple screens (e.g., ACCESS AUTHORIZATION from USER and APPLICATION), specialization (e.g., from the selected APPLICATION down to a particular CATEGORY of alternatives), and instantiation (e.g., from a particular CATEGORY of alternatives down to the actual ALTERNATIVES, indicated by the vertical bar to the right of the decision matrix).

The function of each object type in Fig. 2 can be briefly summarized as follows. A user may choose to work on an MCDM application if he/she is authorized to do so. The user then selects a method for the MCDM session to work on that application. Subsequently, in order to create a decision matrix -- the typical starting point for most MCDM methods -- the user has to define, how decision alternatives and decision criteria are to be derived from the underlying database.

Alternatives are defined in two steps. First, the user selects a subset or category of alternatives to be considered. For example, in a car buying application, the user may be interested only in trucks but not in other types of cars. This step takes one or more database relations as its input to extract from them by selection and join operations a single selected subrelation ("category") on which all further processing will be performed. Second, the user chooses a grouping of database records within this subrelation such that each group constitutes an alternative. For example, the user may be interested only in distinguishing cars by their make and series, but not, e.g., by details such as number of doors, kind of engines, etc.

Criteria are derived from attributes of the database records. In the simplest case, an attribute value can directly serve as a criterion (e.g., maximum speed). However, frequently, the criterion value may involve a function of several attribute values (e.g., average fuel consumption as the average of fuel consumption in the city and on highways). Moreover, whenever alternatives correspond to groups of records rather than to single records, criterion values must be based on aggregate functions over these records (e.g., average, minimum, maximum, forecast for next year).

Finally, the combination of alternative definitions (grouping) and criteria definitions (computations) allows the computation of criterion values for alternatives (CRIT-VALUE) from the underlying database.

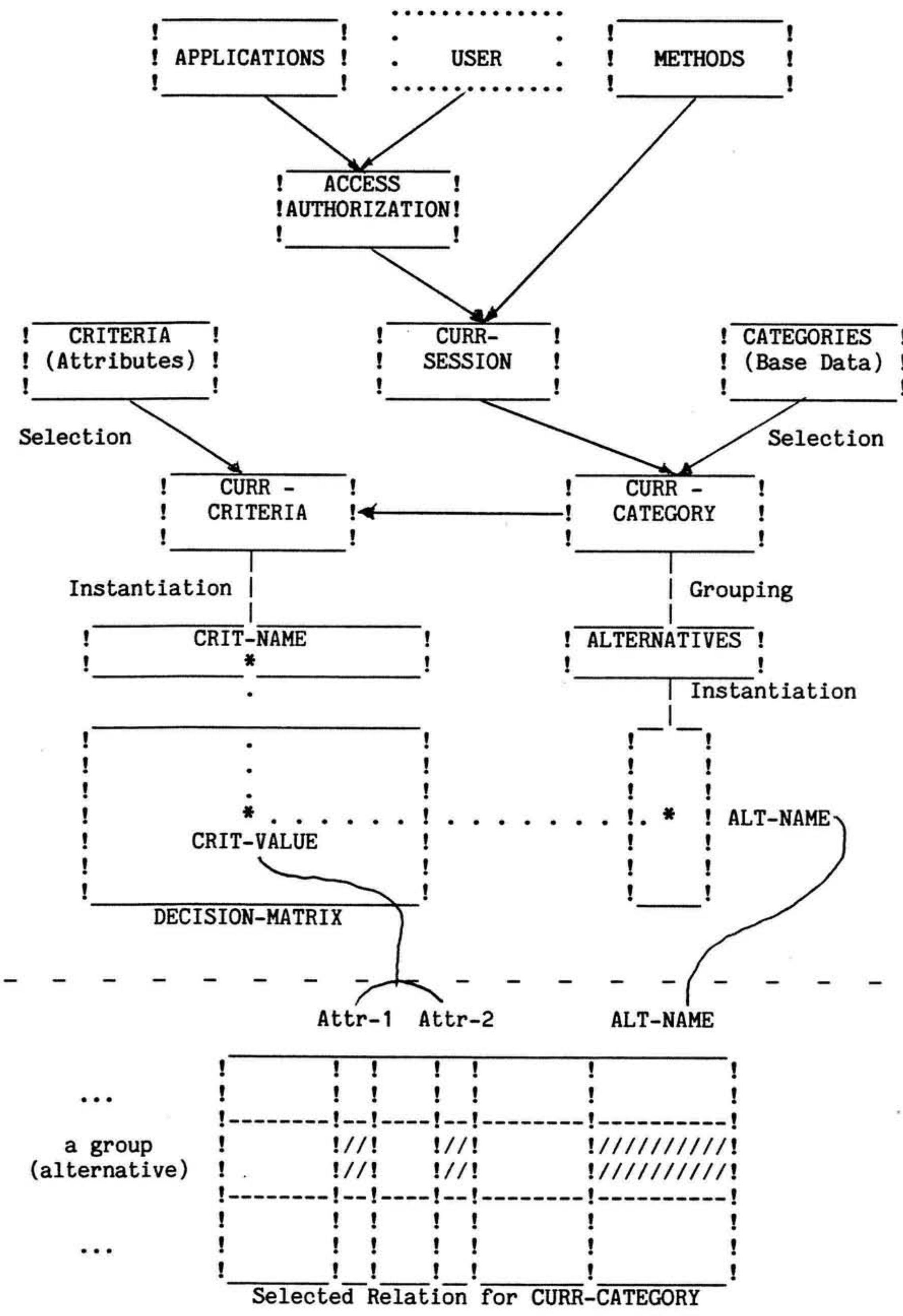


Fig. 2: Abstraction Levels for Model Selection and Data Extraction

4.2 Relational Representation Of The Hierarchy

The proposed abstraction hierarchy can be implemented using an extended relational database system. The extension of the model is relatively small; we only add a few new domain types to capture the semantics of the abstraction process. This allows us to manage metadata like ordinary database data but still permits restrictions on the operations to be performed on such data. The concepts are somewhat similar to those proposed in the area of statistical databases [Shoshani 1982, McCarthy 1982]. New domain types include:

1. Category names and definitions: this is used in the definition of categories, e.g., we may want to introduce a category 'compact', defined by a query:

   ```
   DEFINE CATEGORY compact AS
          SELECT   *
          FROM     car-relation
          WHERE    length >= 10ft AND length <= 20ft;
   ```

   A category is a particular type of view whose definition does only allow a '*' in the SELECT clause (similar to the 'selector' proposed by [Schmidt 1984]). This actually is not an extension of the language power of relational languages since views are available in several DBMS; however, here these data and definitions are stored as field values in relations.

2. Function names and definitions: this is used in the definition of criteria from database attributes [Jarke, 1983]. Over conventional database languages (e.g., SQL), it gives two advantages. First, one can give a name to columns

with aggregate functions; second, the set of aggregate functions is not limited to those offered as built-in functions of the DBMS. The extension of relational query languages by functions with grouping still needs a theoretical foundation, despite the pioneering work of Klug [1982]. We are working on such a framework but an extended discussion is beyond the scope of this paper. As an example, we may define a criterion, space, from underlying database attributes, length and width, in a Pascal-like notation:

```
DEFINE CRITERION space AS
       FUNCTION spfct(rel : car-reltype) : REAL;
       BEGIN
         spfct := SELECT avg(length * width)
                  FROM   rel
       END;
```

where "rel" can be any subrelation of the car-relation (i.e., has the same relation type). Subsequently, we can call this function for an arbitrary category and grouping of alternatives, e.g.,

```
SELECT   MAKE, SERIES, SPACE
FROM     COMPACT
GROUP-BY MAKE, SERIES;
```

Note, that the combination of categories with generalized functions provides a powerful 'generalized view' capability not available in standard database systems.

3. <u>Procedure</u> <u>names</u> <u>and</u> <u>definitions</u>: These are similar to functions, except that they do not return a value but just start the execution of a model. This extension is needed to execute models from the database and is similar to previous work in relational model management (e.g., [Blanning, 1984]).

Bearing these extensions in mind, we can now proceed to define the relations underlying the object types in Fig. 2. The syntax roughly follows SQL, as given in [Date, 1982], with details (such as field lengths) omitted.

The first four relations capture the information required to initialize a session on the DSS.

```
TABLE  APPLICATIONS          ( APP-NAME          (STRING, KEY),
                               APP-DESCRIPTION   (TEXT))

TABLE  ACCESS-AUTHORIZATION  ( APP-NAME          (STRING, KEY),
                               USER-NAME         (STRING, KEY),
                               PASSWORD          (STRING, NOPRINT) )

TABLE  METHODS               ( METH-NAME         (STRING, KEY),
                               METH-DESCRIPTION  (TEXT),
                               METH-PROCEDURE    (PROCEDURE-TYPE) )

TABLE  CURR-SESSION          ( USER-NAME         (STRING, KEY),
                               APP-NAME          (STRING, KEY),
                               METH-NAME         (STRING, KEY),
                               DATE              (INTEGER, KEY),
                               TIME              (INTEGER, KEY) )
```

On the next level, the choice and/or definition of a new category of alternatives is stored.

```
TABLE  CURR-CATEGORY         ( USER-NAME         (STRING, KEY),
                               APP-NAME          (STRING, KEY),
                               METH-NAME         (STRING, KEY),
                               DATE              (INTEGER, KEY),
                               TIME              (INTEGER, KEY),
                               CAT-NAME          (STRING, KEY) )

TABLE  CATEGORIES            ( CAT-NAME          (STRING, KEY),
                               VIEW-DEF          (QUERY-TYPE) )
```

Within each category, the grouping of alternatives and the choice and definition of criteria must be decided. An alternative is defined by keeping a combination of values in given columns of a base relation constant. This partitions the rows into groups. The columns are defined in the relation ALTERNATIVES. For each group of rows, the criterion computation then proceeds as described above.

```
TABLE ALTERNATIVES          ( USER-NAME        (STRING, KEY),
                              APP-NAME         (STRING, KEY),
                              METH-NAME        (STRING, KEY),
                              DATE             (INTEGER, KEY),
                              TIME             (INTEGER, KEY),
                              CAT-NAME         (STRING, KEY),
                              ALT-ATTRIBUTE    (ATTRIBUTE, KEY) )

TABLE  CURR-CRITERIA        ( USER-NAME        (STRING, KEY),
                              APP-NAME         (STRING, KEY),
                              METH-NAME        (STRING, KEY),
                              DATE             (INTEGER, KEY),
                              TIME             (INTEGER, KEY),
                              CAT-NAME         (STRING, KEY),
                              CRIT-NAME        (STRING, KEY) )

TABLE  CRITERIA             ( APP-NAME         (STRING, KEY),
                              CAT-NAME         (STRING, KEY),
                              CRIT-NAME        (STRING, KEY),
                              CRIT-DESCR       (TEXT),
                              MEASURE-UNIT     (STRING),
                              CRIT-FUNCTION    (FUNCTION) )
```

Using the above definitions it is possible to construct a generalized view definition (i.e., query with functions) that generates the decision matrix from the underlying base relation. Depending on how the decision matrix is to be stored, there are several ways to proceed. The method illustrated below is probably the simplest. It assigns one tuple of a relation CRIT-ALT to each criterion column in the decision matrix; essentially, CRIT-ALT is a representation of the join between CURR-CRITERIA and ALTERNATIVES. The ALT-NAME is constructed from the ALTERNATIVES relation and must be the same for all criteria; it simply consists of the list of attribute names that identify an alternative (this will often be a subkey of the selected relation). Thus, the identifier of an alternative is the combination of values in these attributes.

```
TABLE  CRIT-ALT          ( USER-NAME          (STRING, KEY),
                           APP-NAME           (STRING, KEY),
                           METH-NAME          (STRING, KEY),
                           DATE               (INTEGER, KEY),
                           TIME               (INTEGER, KEY),
                           CAT-NAME           (STRING, KEY),
                           ALT-NAME           (LIST-OF-ALT-ATTRIBUTES, KEY),
                           CRIT-NAME          (STRING, KEY) )
```

From CRIT-ALT, the decision matrix is generated as follows. The identifier of the decision matrix is the sixtuple <USER-NAME, APP-NAME, METH-NAME, DATE, TIME, CAT-NAME>. Its column names are: the list of attributes that constitutes an alternative (ALT-NAME) and the list of CRIT-NAMEs associated with the table. Each row contains a unique combination of attribute values in the ALT-NAME columns, and the values computed from CRIT-FUNCTION in each column given by the corresponding CRIT-NAME. In the extended SQL notation, the Query Language Facility therefore produces the following kind of query:

```
DECISION-MATRIX =
        SELECT <ALT-NAME attributes>, <CRIT-NAME 1, ..., CRIT-NAME n>
        FROM   <CAT-NAME>
        GROUP-BY <ALT-NAME attributes>
```

The generalized view processor accepts this query, substitutes the corresponding function definitions from CRITERIA for the criterion names and the view definition from CATEGORIES for the CAT-NAME, and optimizes the resulting query before submitting it to the DBMS or the staging processor. Both the selected category relation and its derived decision matrix are then loaded into the local database (which we assume to be on a microcomputer).

Now, the MCDM method defined in the METHOD relation takes control; during its execution, the user may wish to define new criteria, either manually or by recomputing the decision matrix from the category relation using additional CRITERIA entries. For the sake of brevity, we cannot describe these operations in detail; we just mention at this point that the user has two options: he/she can either change the decision matrix or the underlying selected relation. Only in the latter case, will it, in general, be possible to make the additions to the data permanent -- by writing the selected relation back into the underlying database -- because of the simple nature of the operations allowed for category definition.

5.0 USER INTERFACE GENERATION -- AN EXAMPLE

We now show how model builders can use the abstraction hierarchy of Section 4 to generate a user interface that will allow users to access and run MCDM models. We will show a typical series of screens and give examples of the code required to generate them. In the limited space available many details must be omitted.

A contribution of the proposed GMCDSS is a coupling of the DBMS Query Language with a screen generator. As with other Application Generators, screens can be 'composed' and stored in an off-line database. Additionally, however, one or more windows in the screen can be reserved for displaying database query results. Furthermore, if desired, these can be formatted automatically so that the retrieved tuples appear as menu choices. We distinguish three distinct types of function served by terminal displays: (1) information display, (2) data entry/update and (3) menu choice. Classifying the different screen types according to their primary function and as to whether or not they require a database interaction produces the six basic screen types shown in Table 1.

	No database Interaction	Database Interaction
Information Display	(1) DISPLAY-INFO-SCREEN Help screens, predefined data.	(4) GEN-INFO-SCREEN Display query results.
Data-entry /update	(2) DISPLAY-ENTRY-SCREEN Input values to program variables.	(5) GEN-ENTRY-SCREEN Insert/update/delete/ match database values.
Menu-choice	(3) DISPLAY-MENU-SCREEN Fixed set of choices.	(6) GEN-MENU-SCREEN Variable set of choices determined by DBMS query.

Table 1: Categories of Screens: Command Names and Sample Applications

The commands that will activate each category of screen are shown in capital letters in Table 1. Differentiating the types of screen in this way allows the system to automatically generate the screens and supervise the user interaction thereby reducing programming effort. The model builders can partially predefine all screen types by specifying input and output fields, comments, headers etc. However, the top and bottom sections of the screens are automatically generated by the system and contain application-independent menu choices, help and other information. This varies with the class of screen but provides a measure of standardization and a uniform way of interacting with the system which is application and programmer independent. In the following sample screens, the top line (containing the screen name), and the function keys F1, F2, F3 (bottom of screen) are simplified examples of system-supplied screen sections.

Screens of types (4), (5) and (6) invoke a range of DBMS services while (3) and (6) activate menu-management services. Type (4) and (5) screens are available in some commercial DBMS. With type (4) screens the user can scroll both left and right and up and down and execute searches over the results of the retrieved query (similar to a full screen editor). Type (5) screens are composed simultaneously with the definition of the relations in the database. These screens allow record at-a-time interaction with database relations. The screen fields correspond to relational attributes. In addition to the traditional operations of insert, delete, and update, it is convenient to introduce a 'match' operation which requires the user to input attribute values to be compared to the stored relation. This new command is not only useful for password checking, as shown below, but also for double-checking data entry. Type (6) screens appear to be novel. As discussed below, they provide a simple means for the model builders to construct interfaces for dynamically varying situations.

For all types of screens it is possible for the programmer to write variable values into predefined fields (WRITE .. TO .. command) and to accept information input by the user from predefined fields (READ .. FROM .. command). With screens produced by the GEN-MENU-SCREEN command it is also possible to assign the retrieved values of database fields to program variables (ACCEPT .. FROM .. USING command). We now give some examples of the use of these commands. Note that the dynamically generated sections of the screen will be depicted as lying between ruled lines. Language keywords will be capitalized; variable names, field names and constant values are in small letters.

```
 ______________________________________________________________
| Screen: Access                                               |
|                    MCDSS SYSTEM                              |
|______________________________________________________________|
|                                                              |
|        User-Authorization -- Please enter                    |
|                                                              |
|        User-name     : ______________                        |
|                                                              |
|        Password      : ________                              |
|                                                              |
|                                                              |
|______________________________________________________________|
|                                                              |
| F1 = Help     F2 = Prior screen      F3 = End session        |
|______________________________________________________________|
```

The 'Access' screen provides the first interaction between the MCDSS and the decision-maker. There are two methods of generating such a screen. Using a type (2) screen, the code could be:

```
DISPLAY-ENTRY-SCREEN Access;
    READ User-name-var FROM User-field;
    READ Password-var FROM PW-field;
    IF Password-var NOT IN SELECT Password FROM Access-Authorization
                           WHERE User-name-var = User-name;
        DISPLAY-INFO-SCREEN PW-Violation;
        END-SESSION;
END-DISPLAY-SCREEN;
```

PW-Violation is a type (1) screen. The database query performs a simple table-lookup. If the password does not correspond to that stored for the user the session is ended.

An equivalent method uses a type (5) screen definition:

```
GEN-ENTRY-SCREEN Access;
    MENU Window-1
       MATCH User-name, Password
       WITH  Access-Authorization
       IFNOT DISPLAY-INFO-SCREEN PW-Violation;
             END-SESSION;
END GEN-ENTRY-SCREEN;
```

This method generates the whole window of the screen above (relation name, field names, field lengths and types) from the database. Although it may look more elegant in this particular case, using the type (2) screen buys more flexibility at the expense of increased programming and screen definition effort.

We will suppose that the decision-making session continues by asking the user to select the MCDM application of interest to him/her via the Applic-menu screen below.

```
 _________________________________________________________
| Screen: Applic-menu                                     |
|                    MCDSS SYSTEM                         |
|                                                         |
|_________________________________________________________|
|                                                         |
|  The available Applications are:                        |
|                                                         |
|   1.  Cars                 Car-buying decision          |
|   2.  Homes                Home-buying decision         |
|   3.  Micro-computers      Micro-computer selection     |
|   4.  Travel-packages      Travel-package selection     |
|                                                         |
|   Enter your choice: __                                 |
|_________________________________________________________|
|                                                         |
| F1 = Help     F2 = Prior screen      F3 = End session   |
|_________________________________________________________|
```

The related program segment is:

```
GEN-MENU-SCREEN Applic-menu;
    SELECT  App-name, App-description
    FROM    Application;
    ACCEPT App-name-var FROM App-name USING Choice;
END-GEN-MENU-SCREEN;
```

This is a type (6) screen. The choices in the middle window of the screen, as well as the relation name, are generated from the SELECT command and would vary depending on the current contents of the database. The MENU command performs geometric calculations and provision is made to allow for scrolling of the screen if the information does not fit in the window. The ACCEPT verb causes the assignment of the value of the database App-name field to the program variable App-name-var based on the numeric value input into the 'Choice' field on the screen by the user. Thus if the user types '2', App-name-var gets the value 'Homes'.

```
 ____________________________________________________________
| Screen: Method-menu                                        |
|                       MCDSS SYSTEM                         |
|                                                            |
| 1.  DEFAULT    The system will select a suitable           |
|                method for your application                 |
|___________________________________________________________ |
|                                                            |
|   The available Methods are:                               |
|                                                            |
| 2.  UTA        Assesses additive utility functions         |
|                which aggregate multiple-criteria in        |
|                a composite criterion using the infor-      |
|                mation given by a subjective ranking.       |
|                                                            |
| 3.  ELECTRE    Aggregates weak orders into an outran-      |
|                king relation and produces rankings.        |
|                                                            |
|  Enter your choice: __                                     |
|____________________________________________________________|
|                                                            |
| F1 = Help      F2 = Prior screen       F3 = End session    |
|____________________________________________________________|
```

Method-menu is dynamically generated in a similar way to Applic-menu. The database query associated with this screen is:

```
SELECT Meth-name, Meth-description (40)
FROM   Methods
```

The number in parentheses specifies a limit on the displayed width of a database field and causes the wrap-around shown in the illustration. Note that the DEFAULT choice was pre-specified; the choices in the DBMS window therefore start at 2.

It is now possible to add a new row to the MCDM session relation which tracks the history of user interactions with the system:

```
  INSERT  INTO  Curr-session
         <User-name-var, App-name-var, Meth-name-var, Date-var, Time-var);
```

Here USER-NAME-VAR through METH-NAME-VAR are program variables generated through the above interaction while DATE-VAR and TIME-VAR are system-supplied variables.

```
 _______________________________________________________________
| Screen: Cat-menu                                              |
|                       MCDSS SYSTEM                            |
|                                                               |
|        1.  See the definition of an existing category         |
|        2.  Define your own category                           |
|_______________________________________________________________|
|                                                               |
|       The available Categories are:                           |
|                                                               |
|        3.  Subcompact                                         |
|        4.  Compact                                            |
|        5.  Station-wagon                                      |
|        6.  Trucks                                             |
|                                                               |
|       Enter your choice: __                                   |
|_______________________________________________________________|
|                                                               |
| F1 = Help     F2 = Prior screen       F3 = End session        |
|_______________________________________________________________|
```

This type (6) menu allows the end user to select (or define) the category of alternatives of interest to him/her within the chosen MCDM application. The DBMS query used to generate this screen is:

```
SELECT  Cat-name
FROM    Categories
WHERE   App-name = App-name-var
```

Choice 1 generates a second menu of the same form to allow the user to choose the criterion to be described. Choice 2 will lead to a type (5) insert screen for the relation CATEGORIES.

```
 ____________________________________________________________
| Screen: Crit-menu                                          |
|                    MCDSS SYSTEM                            |
|                                                            |
|     1. See the definition of an existing criterion        |
|     2. Define an aggregate criterion                       |
|____________________________________________________________|
|                                                            |
|     The available Criteria are:                            |
|                                                            |
|     3. Price                                               |
|     4. Maximum speed                                       |
|     5. Horse-power                                         |
|     6. Number-of-doors                                     |
|     7. Number-of-seats                                     |
|     .....more.........                                     |
|    Enter your choice: __   or  F10 for other choices       |
|____________________________________________________________|
|                                                            |
| F1 = Help     F2 = Prior screen       F3 = End session     |
|____________________________________________________________|
```

In this menu, the end user is asked to select (or define) the criteria that will be evaluated for every alternative of the chosen category. The existence of other choices that cannot fit in the available window is indicated.

Choice 2 will require the definition of a new criterion in a type (5) insert screen. In the simplest case, this will just mean renaming an existing attribute or criterion. Otherwise, the user must define a new function which may require a relatively high level of skill. An intermediate alternative would be to provide a type (3) menu with available standard functions. The corresponding code for the type (5) screen would be:

```
GEN-ENTRY-SCREEN Criterion-def;
    INSERT App-name-var, Cat-name-var, Crit-name,
           Crit-Descr, Crit-function
    INTO   Criteria;
```

The last step in the data extraction is the specification of alternatives, i.e., the grouping of the tuples in the selected category relation. Since the GROUP-BY clause can have more than one attribute this requires the selection of a group of attributes from a type (6) screen displaying as menu choices the list of attributes of the selected category relation. The system has to check for duplicate choices and for disjunctness with the set of attributes underlying the selected criteria. At this stage of the preparation process, the GMCDSS will generate the relation CRIT-ALT associating the alternatives of interest to the user with the criteria. The further procedure is as described in section 4.

6.0 CONCLUSIONS AND FUTURE WORK

In this paper, we developed several database-centered methods aiming at an improved integration of database, model and dialog management in DSS. Using the example of multiple criteria decision making, we first introduced an abstraction hierarchy the user can employ for a stepwise selection and refinement of the problem to be solved. Second, some extensions to the relational model of data were introduced that permit the mapping of the hierarchy into an enhanced relational database. Finally, it was demonstrated how this database in turn can be utilized to substantially facilitate screen management in the Dialog Manager, using a taxonomy of screen generation procedures.

Due to the large number of concepts covered in this paper, many details remain to be worked-out in depth. In particular, two issues need further research. First is the development of a sound theoretical foundation for the functional enhancements of the relational data model introduced in section 4; syntactical details and the efficient evaluation of generalized views must be studied. Second is the detailed design and implementation of the screen manager outlined in section 5; in particular, the relationship of the proposed data-driven menu generator with windowing capabilities, and the relationship between static declarations of screen formats and dynamic activation of stored formats in a program will be investigated.

It is our hope that the combination of the proposed concepts will finally lead beyond the well-known architectural paradigms of DSS that simply add-up subsystems, towards a kernel DSS architecture that is still able to communicate with existing sources of data and models but has its own 'personality'.

References

Blanning, R.W., "Language Design for Relational Model Management", in: Chang, S.K. (ed.), Management and Office Information Systems, New York: Plenum Press, 1984, pp. 217-235.

Bui, X.T., "Building Effective Multiple Criteria Decision Models: A Decision Support System Approach", Systems, Objectives, Solutions, Vol. 4, No. 1, pp. 3-16.

Date, C.J., An Introduction to Database Systems, Third Edition, Reading, Mass.: Addison-Wesley, 1982.

Ignizio, James P., Goal Programming and Extensions, D. C. Heath and Company, Lexington, Mass., 1976.

Jacquet-LaGreze, E., and Siskos, J., "Assessing a Set of Additive Utility Functions for Multiple Criteria Decision Making -- the UTA Method", European Journal of Operational Research, Vol. 10, No. 2, pp. 151-164.

Jarke, M., "Problems of Database Usage in Health-Economic Cost-Effectiveness Studies", Proceedings 12th Annual DGOR Congress, Mannheim, West Germany, Heidelberg: Springer-Verlag 1983 (in German).

Jelassi, M.T., Jarke, M., and Stohr, E.A., "Designing A Generalized Multiple Criteria Decision Support System", Proceedings VIth International Conference on Multiple-Criteria Decision Making, Cleveland (Ohio), to be published by Springer-Verlag (1985).

Keeney, R.L., Raiffa, J., Decisions with Multiple Objectives: Preferences and Values Trade-Offs, New York: John Wiley 1976.

Klug, A., "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions", Journal of the ACM, Vol. 29, No. 3 (1982), pp. 697-717.

McCarthy, J.L., "Metadata Management for Large Statistical Databases", Proceedings 8th International Conference on Very Large Data Bases, Mexico City, September, 1982, pp. 234-243.

Roy, B., "Classement et choix en presence de point de vues multiple (la methode ELECTRE)", R.I.R.O., Vol. 2, pp. 57-75.

Saaty, T., The Analytic Hierarchy Process: Planning, Priority, Allocation, New York: McGraw Hill, 1980.

Schmidt, J.W., "Database Programming: Language Constructs and Execution Models", in U.Ammann (ed.): Programmiersprachen und Programmentwicklung, Heidelberg: Springer-Verlag 1984, pp. 1-26.

Shoshani, A., "Statistical Databases: Characteristics, Problems, and Some Solutions", Proceedings 8th International Conference on Very Large Data Bases, Mexico City 1982, pp. 208-222.

Smith, J.M., and Smith, D.C.P., "Database Abstractions: Aggregation and Generalization", ACM Transactions on Database Systems, Vol. 2, No. 2 (1977), pp. 105-133.

Sprague, R., and Carlson, E., Building Effective Decision Support Systems, Englewood Cliffs, NJ: Prentice Hall, 1982.

Stohr, E.A., and White, N.H., "User Interfaces for Decision Support Systems: An Overview", International Journal of Policy Analysis and Information Systems, Vol. 6, No. 4 (1982).

Zeleny, M., Multiple Objective Decision Making, Reading, Mass.: Addison-Wesley, 1982.