ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

B Novikov, University of St Petersburg, Russia and J W Schmidt, University of Hamburg, Germany. (Eds)

Advances in Databases and Information Systems, Moscow 1996

Proceedings of the International Workshop on Advances in Databases and Information Systems (ADBIS '96). Moscow, 10-13 September 1996

User Modelling for Conceptual Database Design Based on an Extended Entity Relationship Model: A Preliminary Study

Margita Altus

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

User Modelling for Conceptual Database Design Based on an Extended Entity Relationship Model: A Preliminary Study

Margita Altus

Computer Science Institute

Brandenburg Technical University of Cottbus

Research Group Database and Information Systems

Abstract

We introduce a new point of view into user modelling by analysing systematically the *task* of a database designer. The database designer accomplishes the design process using design steps. Design primitives are possible values of design steps. The main idea of the paper is to support an arbitrary *user* with an evaluation function and a finite set of application independent design primitives. A mechanism based on the belief function of the Dempster-Shafer calculus can be used to solve different problems: finding the most appropriate design strategy w.r.t. a specific designer and application, determining a plausible design primitive for the next design step. Finally, an Explanation Component can use this mechanism to guide the designer in an efficient way. The User Guidance Tool is sketched, whereby some aspects of context-sensitivity of the Explanation Component are shown in more detail. The user interface design is presented by an overview of the functions which are included in the RADD system.

1 Introduction

The task of conceptual database design is to determine if there is a *plausible* relationship between a concept of the design model (design primitive) and the concept of the application domain. A meta-model of conceptual database design based on plausible inference suggests that techniques should be found for combining multiple sources of evidence (e.g. properties of the designer and the application) or search strategies into an overall assessment of a design primitive relevance rather than attempting to pick a single design strategy. In this paper, our approach to plausible inference for conceptual database design is explained and some preliminary experiments designed to test this approach are described.

We use a plausibility function which is closely related to the belief function of the Dempster-Shafer calculus (DSc). This kind of modelling the designer's beliefs and goals is a step towards the formalization of a user model for conceptual database design. The DSc provides the possibility of combining *data* of the user model space and other knowledge, usually considered as *evidence*. Furthermore, all data used as evidences has to be considered as uncertain and imprecise. The combination of uncertain and imprecise data known as evidences supports a new database model, [10]. In addition, at present the DSc is being investigated by researchers and practitioners, e.g. in data mining, [6].

Only a very small portion of the design systems contains expertise that was elicited from experienced, or expert, database designers, or contains mechanisms that allow an expert to update the system's knowledge base ([26], p.98).

In recent years, a considerable amount of research has been focused on understanding, formalizing, and automating design, leading to the development of various expert system tools (e.g. Silverrun, MetaEdit, InfoModeler, ERDRAW, DBMain, Bonapart, DDEW [17, 27, 14, 24]). These systems have to be characterized as assistants to database designers. They are not being developed to replace a human expert designer. Database design expertise is used to reduce the search space of the design process.

1.1 The Design State Space

Methods for searching are to be found at the core of many AI systems, e.g. [28]. The database designer must also search along constrained paths through intricate networks of knowledge, design states or conditions to find the relevant

design information or to reach the design goal position. The goal of database design is to find a database design that meets several quality criteria, e.g. efficiency, minimality, readability, understandability, simplicity, expressiveness, self-explanation [9], the best structure with respect to the behaviour and implementation of the database. Thus, the database design process is a complex search for the most appropriate design meeting as many quality criteria as possible.

The set of all possible states for a database design problem together with the relations involving states implied by the transformations or operators is called the *state space of the design process*. In general, the designer may move from one state to another. But, experienced designers in particular use their design knowledge and apply design heuristics, e.g. known from the database design theory as the different kinds of normalization [9, 23]. Normalization is achieved by the decomposition of concepts. Nevertheless, normalization is always achieved by introducing a separate concept for each object of the application domain ([9], p.158). In this way, experienced designers apply other heuristics like different kinds of abstraction techniques to find separate semantic units of the application domain and to map the concepts of the domain to the concepts of the design model.

Furthermore, heuristics are used at the conceptual level for optimization - of course, optimization is not a conceptual issue. The term optimization describes procedures for improving the performance of a system ([15], p.192).

Therefore, classical *search methods* like depth-first or breadth-first search are not adequate for the database design problem. These search methods are “blind” in the sense that they use exhaustive approaches. Instead, each design state is evaluated by the designer using different kinds of knowledge (e.g. global and specific domain knowledge, design knowledge) and then s/he decides which design state would be the best successor. This kind of designer applies a heuristic search method like the *A** algorithm using an estimation function. An arbitrary user of a database design environment can be supported by implementing such search techniques providing *evaluation functions* and *design primitives*. The evaluation function estimates what the most *plausible* next design step w.r.t. the design goal is. Design primitives are used as *abstract operators* for moving from one design state to another. In the following, the term design strategy is used as a synonym for search method.

There are some special cases of design strategies, known as the classical design strategies: top-down, bottom-up, mixed, inside-out and modular design strategy [9].

Each design strategy is characterized by different approaches: *The design direction*: the bottom-up design uses composition operations, the top-down design uses decomposition operations. *The control of the design*: Inside-out design strategies use a neighbourhood function to select the next concept. *The degree of modularity of the design*: The mixed design starts with the skeleton design and is then refined; this is analogous to modular design; the view-oriented design is a special case of the modular design. Therefore, a design strategy is composed of different approaches. The database design process is more a *process of construction* than a process of mapping knowledge of the application. The design process is accomplished by design steps. Each design step is based on a *design primitive* and may require consistence checks. Design primitives are mainly characterized by the following operations: *composition*, *decomposition* and *extension*. [3]

In summary,

- each design step is based on a design primitive,
- design strategies are composed of different approaches,
- single design steps are recursively described by design strategies.

The analysis of several designer groups shows that the selection of a design strategy depends on:

- the skills, the capabilities, the properties, the knowledge and the experience of the designer,
- the underlying design model,
- the complexity of the application, especially the structure of the application, the degree of modularity and the complexity of semantics and operations,
- the size and experience of the designer team.

1.2 The Design Complexity Problem

Besides the number of design objects as the attribute types, entity types and relationship types, the complexity of the design is determined by the design complexity of integrity constraints and the design complexity of the application operations. The design complexity, especially for large applications and for a design process in designer teams, requires special support through a design workbench. Thus, the quality of the design result depends primarily on the professionalism and experience of the designer and on the support of appropriate CASE-Tools.

One approach would be to consider the design problem as a construction task. The design construction process is a sequence of design steps and each design step is an operator that transforms one design state into another. With an average length n of the sequence of design steps and an average number m of alternative design steps the upper bound of the time complexity of the search function is $\mathcal{O}(m^n)$. A conceptual database design assistant should support the designer during the whole design process with the aim to reach the design goal with a relatively small number of design steps (m) and design primitives (n). Nevertheless, it is obvious that changes of the design strategy may result in changes of the set of alternative design steps and increase the design complexity. Furthermore, it is known that designers use different design strategies and, even more, they change the strategy in dependence on the design state reached.

1.3 Required Tool Support

The system RADD (Rapid Application and Database Development, [2]) developed in groups in Cottbus, Dresden, Hamburg, Kuwait, Münster, and Rostock [7, 30] does not require the user to understand theory, implementational restrictions and programming problems in order to design a database schema. A novice designer can create a database design successfully using the system. These tools are based on an extended entity-relationship model. The entity-relationship model has been extended to the *Higher-order Entity-Relationship Model* (HERM, [30, 31]) by adding structural constructs and using integrity constraints and operations. Different database design methodologies have been developed based on the HERM approach.

The RADD design workbench is *adaptable* to a designer and his/her design techniques. It supports the designers in the choice of a design strategy according to his experience and abilities and it comprises mechanisms for consistence checks of each design decision. Furthermore, it captures design consultancy for the whole design process. The first design step may be accomplished in the natural (German) language [1]. During a moderated dialog a skeleton of a design is developed. The whole design information represented in the extended Entity Relationship Model (HERM) can be specified in an integrated way. That includes besides the representation of the structure, the representation of integrity constraints, operations, views and queries. The design can be transformed by the system into different languages for the logical and physical representation of a certain database management system and the system uses in dependence upon the chosen DBMS tuning techniques which are included in an optimization component ([2]).

The remaining sections are divided as follows. Section 2 presents the subgoals of user modelling in design environments and the acquisition and representation of different pieces of user information. To make the paper self-contained, Section 3 then motivates the application of the Dempster-Shafer calculus for the computation of the next design step, the initial design strategy and topics in an explanation component. Section 4 then gives a general description of the user guidance tool and presents meta-information for the primary key problem. Section 5 describes briefly the function which are included in the RADD system. The paper concludes in Section 6 implementational issues.

2 User Modelling

Adapting to the background and interests of a person requires information about the beliefs and the goal of the user : in effect, a user model.[8]

The *user* of the database design environment RADD is classified regarding his properties, his capabilities, his preferences (kinds of input, output and dialog) and his system knowledge, his application knowledge and his knowledge about design concepts and design strategies, and he can be supported with respect to this *classification*. The user

models are constructed on the basis of the interactions with the user. The goal of the *User Modelling Component* in RADD is composed of five subgoals:

1. The design interface should be adapted to the designer's preferences.
2. The design strategy should be adapted to the designer, especially to his design expertise.
3. The derivation of a plausible design primitive for the next design step is based on different sources of evidences.
4. The explanation and discussion of design decisions and design errors should be accomplished in a context-sensitive and user-oriented style.
5. The design information should be shown in a "user-friendly" way.

Acquisition of the User Information. The user's knowledge and actions are analysed. The user analysis is divided into two parts: the direct analysis (user's answers in an interrogation) and the indirect analysis. The indirect analysis evaluates each user's action in the design environment. There are two *kinds of user action*: using a design primitive (1), activating any tool of the RADD system (2) - strategy advisor, concept advisor, design consultant, consistence checker, error locator, schema editor, controller, behaviour estimator, semantic acquisition design assistant ([1]), natural language moderator, examples tutor, context-sensitive user-oriented help-system. The first type of analysis refers to the *explicit* and the second type of analysis to the *implicit acquisition* of user model information.

Representation of User Characteristics. Individual users are represented by a collection of frames. General frames store user-specific information which is long-term static information with respect to the user classification. Action frames store information on the first kind of user action. This kind of user action results in the execution of a schema transformation which is based on one specific design primitive. Each design primitive has a graphical representation and implementation based on graph rewriting rules, [5]. Therefore, action frames so-called *user profiles* contain the information about one design step and can be considered as small design-state-dependent patterns of the user behaviour.

3 Measures of Belief vs Probabilities

The *evidences* for composed database design strategies, atomic design primitives and specific explanation needs are interpreted as partially subjective valid subinformation since the vagueness lies per definition in the individual designers who are going to use a specific design strategy, to apply a design primitive and to consult topics (nodes, links) that refer to hypertext documents.

The use of the Theorem of Bayes has to be regarded as inappropriate for our problem since we have to manipulate measures of belief instead of probabilities. We have to distinguish the *state of ignorance* and uncertainty.

For example, for a given design strategy s_i , S_i represents the proposition "The design strategy s_i is the initial design strategy". We know that it does not always hold that $P(S_i) + P(\neg S_i) = 1$. Suppose that some designer $\mathcal{A}$ does not know what a design strategy is and how the design strategy s_i is characterized. We can not really say that the designer $\mathcal{A}$ believes the proposition S_i if s/he has no idea what it even means or s/he has at most some global

Table 1: Example: The set of alternative design strategies

	d_1 direction	d_2 control	d_3 modularity	d_4 representation
S_1	top-down	semantic neighbourhood	view-oriented	graphical
S_2	bottom-up	uncontrolled	conceptual	graphical
S_3	top-down, bottom-up	skeleton-driven	modular	graphical
S_4	top-down, bottom-up	semantic neighbourhood	conceptual	graphical
S_5	top-down, bottom-up	semantic neighbourhood	modular	graphical, logical, examples

knowledge in design issues and s/he is not familiar with the details. Neither is it precise to say the designer $\mathcal{A}$ believes the negation of the proposition S_i . Denoting the designer's degree of belief using $B(S_i)$, it is reasonable to assign both $B(S_i)$ and $B(\neg S_i)$ a value 0.

Regarding the set of design primitives we can formulate similar propositions, e.g. for the design primitive 'Generalisation' p_j , D_j represents the proposition "The design primitive p_j is appropriate for the next design step". The degree of familiarity of the designer $\mathcal{B}$ with the 'Generalisation design concept' can be expressed by denoting the designer's degree of belief using $B(D_j)$. The same holds for a database design environment that incorporates design knowledge. It is obvious that the occurrence of any new evidence, in relation to database design issues, reduces the state of ignorance.

Furthermore, we need a solution that supports the accumulation of uncertain evidences and the grouping of different design strategies, design steps and topics, respectively. Elements of groups may have the same evidence values since sometimes a differentiation may not be possible.

The Dempster-Shafer calculus is a system for manipulating degrees of belief which is more general than the Bayesian approach and allows a more precise kind of inference from uncertain evidence ([28], p.272).

As the theory of probability, the Dempster-Shafer calculus deals with the possible values of an unknown variable. The set of possible values for a given variable is called the *universe* or the *frame of discernment*, and is usually (but not always) considered to be finite ([28], p.272). For example, in the case of finding the initial design strategy s_i , the sample universe in Table 1 contains five elements (hypotheses). Each row represents one proposition S_i that is based on sub-propositions d_i (approaches/ dimensions composing a design strategy, [30]). The application of the mechanisms based on the DSc w.r.t. this sample universe are considered in more detail in [4].

4 The User Guidance Tool

The User Guidance Tool comprises mechanisms for a consistent development of schemata, for design strategy support, customization and user adaptation. Furthermore, it includes a tutorial and explanation component. The consistent development is internally enforced. Nevertheless, some functions enable the users to explicitly check the consistence. Using a semantic data model the system generates many constraints for consistence enforcement. Therefore, they must be efficient and reflect user's expectation. Hence, work was necessary to find constraint hierarchies, whereby the hierarchies describe a preferential order in which to satisfy constraints. The design strategy support is closely related to the consistent development of a schema and is based on the decomposition of the design process in design steps.

In this section the main ideas of the tutorial and explanation component of the RADD system are sketched. The tutorial component is based on a set of design examples (HERM schemata). The concepts of the HERM methodology are explained by sample schemata comparing similar design situations in different sample schemata for the same design concept (e.g. cardinality constraints, mandatory and optional relationship types; 1:1, M:N, 1:N relationships, generalization, specialisation, aggregation, any design primitives).

The mechanism based on the plausibility function presented in section 3 can be adapted to the selection of a specific online-help support, e.g. hypertext document, combined with design examples from the tutorial component. This kind of context-sensitive online-help support is based on evidences like the kind of error, the design primitive which indicated an error, application knowledge (e.g. a certain subdomain: "Person") and properties, skills, expertise and knowledge of the designer. Therefore, the elements of the set of alternatives (the universe) are terms which refer to hypertext documents. On the one hand, the result of the evaluation of the different pieces of evidences is shown in

Table 2: The order function

$$order(type_i) = \begin{cases} 0 & : \text{iff } type_i \text{ is an entity type} \\ order(c_k) + 1 & : \text{iff } \omega \end{cases}$$

with $\omega = \exists c_k \forall c_j : component(type_i, c_k) \wedge component(type_i, c_j) \wedge (order(c_k) \geq order(c_j))$

Table 3: Meta Schema predicates, sets and function

Predicate, set or function	meaning	possible values
$component(type_i, c_j)$	defines c_j to be a component of $type_i$	true, false
$order(type_i)$	a recursive function defined in Tab. 2	≥ 0
$PrimaryKeyAttributeType(type_i)$	defines the primary key of $type_i$	a set of attribute types
$KeyAttributeType(type_i)$	defines a candidate key of $type_i$	a set of attribute types
$PrimaryKey(type_i, c_j)$	defines c_j to be a component and to be included in the primary key of $type_i$	true, false
$pkcA(type_i, c_j)$	defines c_j to be a component with max. participation 1 in $type_i$ and to be included in the primary key of $type_i$	true, false
$pkcB(type_i, c_j)$	defines c_j to be a component with max. participation > 1 in $type_i$ and to be included in the primary key of $type_i$	true, false
$npkcA(type_i, c_j)$	defines c_j to be a component with max. participation 1 in $type_i$ and to be included in the primary key of $type_i$	true, false
$npkcB(type_i, c_j)$	defines c_j to be a component with max. participation > 1 in $type_i$ and to be included in the primary key of $type_i$	true, false
$comp(type_i, c_j)$	defines for entities of the component c_j the minimal and maximal participation in $type_i$	true, false (0,1), (1,1), (1,.), (0,.), where the 'dot' is a value ≥ 2

a list of *terms (topics) of interest* whereby each of them refers to an appropriate document and are proposed by the system for consulting. On the other hand, the evaluation of different pieces of evidences, especially user's preferences and expertise, results in a user specific kind of presentation of the information that is used for explanation, e.g. long versus short; textual, graphical, formal, or sample based explanation.

There are more specific pieces of evidences w.r.t. the design model. We use the primary key consistence check for relationship types to demonstrate some aspects of context-sensitive explanation.

We borrow the three cases described in [9] and extend these cases to n-ary relationship types. Let R be a n-ary relationship type of order i among entity or relationship types $E_1, \dots, E_n$ of order $\leq i - 1$; let $m_1, \dots, m_n$ denote the maximum of the complexity of $E_1, \dots, E_n$ in R .

1. *One-to-one relationship*: Entity/ relationship types $E_1, \dots, E_n$ with $m_1 = \dots = m_n = 1$. The primary key or any candidate key is incorporated in the primary key of R arbitrarily selected *either* from E_1 or ... or from E_n .
2. *One-to-many relationship*: Entity/ relationship types $E_1, \dots, E_i, E_{i+1}, \dots, E_n$ with $m_1 = \dots = m_i = 1, m_{i+1} = \dots = m_n = p$ ($p > 1$). The primary key or any candidate key is incorporated in the primary key of R arbitrarily selected *either* from E_1 or ... or from E_i .
3. *Many-to-many relationship*: Entity/ relationship types $E_1, \dots, E_n$ with $m_1 = \dots = m_n = p$ ($p > 1$). The primary key or any candidate key is incorporated in the primary key of R from *all* E_i whereby $1 \leq i \leq n$.

With respect to the three cases we can define the primary key consistence condition, design an algorithm for the primary key consistence check and construct a default rule. If a primary key is not explicitly declared then the rule in Tab. 4 is applied to derive a primary key for a relationship type.

The subtask *primary key declaration for relationship types* captures three problems for the designer. First, s/he has to check the uniqueness of the key candidate. Second, s/he has to check the minimality property of the key

Table 4: Default rule for the determination of a primary key (relationship types)

```

IF  $order(type_i) > 0$ 
THEN
  IF  $PrimaryKeyAttributeType(type_i) = \emptyset$ 
  THEN
    IF  $\exists c_j : component(type_i, c_j) \wedge comp(type_i, c_j) = (1, 1)$ 
    THEN  $PrimaryKeyAttributeType(type_i) := KeyAttributeType(c_j)$ 
    ELSEIF  $\exists c_j : component(type_i, c_j) \wedge comp(type_i, c_j) = (0, 1)$ 
    THEN  $PrimaryKeyAttributeType(type_i) := KeyAttributeType(c_j)$ 
    ELSE  $\forall c_j : component(type_i, c_j) \wedge$ 
           $((comp(type_i, c_j) = (1, .)) \vee (comp(type_i, c_j) = (0, .))) \wedge$ 
           $PrimaryKeyAttributeType(type_i) := KeyAttributeType(c_j)$ 
    END
  END
END

```

candidate and finally, w.r.t. the physical design, especially the index problem, s/he should find the shortest key. This consistence check requires different pieces of information: the complexities (cardinalities) of the components w.r.t. $type_i$, the primary key or candidate keys of the components that are included in the primary key of $type_i$, and finally, the primary key attribute types which are directly assigned to $type_i$.

There are five cases where inconsistency can be automatically discovered. An appropriate explanation requires to further subdivide these cases in subcases:

Case1 : $type_i$ has *at least one* component c_j where the complexity constraint (1) holds, but there is *not any* of these components included in the primary key of $type_i$. In addition, there is not any attribute type of $type_i$ declared to be included in the primary key of $type_i$. This case is represented by the expression (3).

Case2 : $type_i$ has *exactly one* component c_j where the complexity constraint (1) holds and c_j is included in the primary key of $type_i$. In addition, *either* $type_i$ has *at least one* component c_k where the complexity constraint (2) holds *or at least one* attribute type of $type_i$ is included in the primary key of $type_i$. This case is represented by the expression (4).

Case3 : $type_i$ has *at least two* components c_j where the complexity constraint (1) holds and *all* these c_j are included in the primary key of $type_i$. This case is represented by the expression (5).

Case4 : $type_i$ has *not any* component c_j where the complexity constraint (1) holds. In addition, there is not any attribute type of $type_i$ is included in the primary key of $type_i$, but there are *at least one* component c_k where the complexity constraint (2) holds and c_k is *not* included in the primary key of $type_i$. This case is represented by the expression (6).

Case5 : $type_i$ is a unary relationship type with one component c_j , where the complexity constraint (2) holds and c_j is included in the primary key of $type_i$. In addition, there is *not any* attribute type of $type_i$ included in the primary key of $type_i$. This case is represented by the expression (7).

In summary, design error occurrence can be accumulated using Dempster's Rule of Combination to determine the most plausible kind of online-help, e.g. a list of terms referring to hypertext documents which should be consulted in the context of the user specific design errors. A design error which occurred very early in the history of user actions can not be removed, but it is less relevant to the current design situation and the degree of explanation that is appropriate for the current designer. Therefore, the explanation component captures properties of a dynamic context-sensitive online-help system.

Table 5: Expressions for the primary key problem

$(comp(type_i, c_j) = (0, 1)) \vee (comp(type_i, c_j) = (1, 1))$	(1)
$(comp(type_i, c_k) = (0, .)) \vee (comp(type_i, c_k) = (1, .))$	(2)
$(pkcA = \emptyset) \wedge (npkcA \neq \emptyset) \wedge (PrimaryKeyAttributeType(type_i) = \emptyset)$	(3)
$(pkcA = 1) \wedge ((pkcB \neq \emptyset) \vee (PrimaryKeyAttributeType(type_i) = \emptyset))$	(4)
$(pkcA > 1)$	(5)
$(pkcA = \emptyset) \wedge (npkcA = \emptyset) \wedge (PrimaryKeyAttributeType(type_i) = \emptyset) \wedge (npkcB \neq \emptyset)$	(6)
$(pkcB = 1) \wedge$	
$(pkcA = npkcA = npkcB = PrimaryKeyAttributeType(type_i) = 0)$	(7)
$pkcA = \{c_j pkcA(type_i, c_j)\}, pkcB = \{c_j pkcB(type_i, c_j)\},$ $npkcA = \{c_j npkcA(type_i, c_j)\}, npkcB = \{c_j npkcB(type_i, c_j)\}$	

Design errors based on inconsistency are only one half of the story. There may be design errors that are not caused by inconsistency, uncertainty or ignorance. These kinds of errors are based on misconceptions. The schema is correct, the designer's knowledge about the current domain concept is certain, but s/he has wrong assumptions about the design concept applied. Therefore, these kinds of misconceptions have to be detected as early as possible and the designers have to be supported with an appropriate explanation. In conclusion, the error analysis approach is originally located in the area of intelligent tutorial systems based on student models [21, 18]. Hence, to create a working user model correctly applied design primitives, never applied design primitives together with the list of errors have to be evaluated.

5 The Schema Editor

To assist users in the specification of HERM schemata the RADD Schema Editor was developed based on a graphical kernel editor known as Graph^{Ed} [16]. The Schema Editor is a window-based tool which enables the users to create, display and manipulate a schema graphically. At the top of the Schema Area there is the menubar, Fig. 1. This area includes nine main utilities provided by the RADD system. These are DataDictionaryFile (short: DDFile), HERM-Concepts, DesignConsultant, DesignTools, BehaviourEstimation, Translator, GraphTools, ExampleTutor, Help. Each of these utilities has a pull-down menu that contains items which refer to functions of the system or other pull-down menus. A pup-up menu with the same structure as the menubar can be shown at each place of the Schema Area.

The DDFile Menu provides filing options, one option for the schema area creation and one option to exit to the operating system.

The ERConcepts Menu consists of several functions related to the design concepts. The user can create an ER-index of varying extent, s/he can set, show in the schema graph or list role names, primary or candidate keys, complexities (classical or average cardinalities) or edit the attribute type structure, Fig. 1. Each function includes check points for consistence.

The DesignConsultant Menu contains three kinds of groups of functions. The first group of functions supports the user with the possibility of checking explicitly the correctness of the current schema at any time. The second group of functions enables the users to ask for design strategy support ('Next Design Concept?', 'Next Design Step?'). The last group of functions allow the users to check the current schema w.r.t. simplicity and expressiveness.

The DesignTools Menu comprises more comprehensive functions implemented by different tools. These tools are accessible in an easy and integrated way. The users can switch between the tools as they can select any arbitrary function of the RADD system. The 'Strategy Advisor' enables the users to determine the initial design strategy (top-down, bottom-up, mixed, inside-out, modular or any derivate of them) using information about the designer and the application. The 'NLI Moderator' allows the designer to express his/ her design information about the application on the basis of the German language and the RADD system is able to extract structural, semantic and operational specification of an application using a moderated dialog [1]. Using sample relations the designer is supported by an

Figure 1: Schema editor and attribute type editor

informal and efficient approach for obtaining semantic constraints implemented by the 'Design By Examples' utility [1]. The 'Concept Advisors' allows the designer to describe the next concept of the application using structured goal-driven dialogs. The 'Customizer' enables the users to change some default values of parameters for the RADD system, e.g. extent of output. Using the result of the 'Strategy Advisor' tool the users are supported by the 'Compose Strategy' utility to tune the properties (dimensions) of the chosen design strategy like the direction, control, degree of modularity.

The BehaviourEstimation Menu consists of a tool that can estimate the complexity of the behaviour during database operation.

The Translator Menu captures three critical functions for setting up a database. The 'SQL Translator' starts a frame to support the users with the function that physically creates a database based on the optimized HERM schema. The 'DBPL Translator' supports the users with two start-up frames (Load or Store) which enable them to set some parameters for the HERM-DBPL gate and start the translator (Load or Store). The 'DBPL User Support' provides the users during the translation of a DBPL program in a HERM schema with short dialogs to find unresolved parts of the schema.

The GraphTools Menu supports the users with several functions to manipulate the schema graph in the schema area *without* changing any design information: Redraw all; resize the node shape (smaller, larger); show grid on/off; node statistics; set the current schema graph on grid points; fit node to the label text; layout-transformer of the schema graph : rotate to the left, rotate to the right, change the top to the bottom, change the left to the right; fit window to the

graph; zoom in, zoom out; schema graph layout algorithms: spring-embedder, DAG-layout. All these functions are imported from the graphical kernel editor Graph^{Ed} [16].

The *ExampleTutor Menu* supports the designers explicitly with a list of HERM schemata designed in earlier sessions. The activation of one example triggers the ‘Explanation Component’ with a list of terms referring to hypertext documents which are related to the selected example and proposed for consulting, and the ‘Data Dictionary File’ of the selected example is loaded by the Schema Editor.

The *Help Menu* provides the users of the RADD system with two kinds of hypertext systems, namely a context-sensitive user-oriented online help system and the ‘Netscape Browser’ based on a selected www address.

6 Implementational Issues

The implemented list of design primitives includes all primitives developed for decomposition and composition of attribute types ($p_0 - p_{12}$). Furthermore, the design primitive for entity type generation (p_{19}), the design primitive for the extension of relationship types with component types or attribute types (p_{16}) and finally the design primitive for the initialization of a database design module (initUnit, p_{22}) are implemented. With respect to their identifiers (p_i) the complete set of design primitives ($p_0 - p_{22}$) is described in [5].

The prototype of the RADD system is running on a SUN Workstation with the OpenLook interface. The Schema Editor (the extensions to Graph^{Ed}: 28,000 lines C source code,) and the DBPL-HERM gate (17,000 lines C source code) are implemented in C. The strategy advisor and concept advisor are implemented in SUN’s Modula2. The concept advisor includes 3,000 lines source code, the strategy advisor includes 5,000 lines source code, the Natural Language Interface is implemented by Quintus-Prolog and the Behaviour Estimation Component is based on the language Standard ML. The Design-By-Examples tool is implemented in SUN’s Modula2.

7 Acknowledgement

The research described here was performed when the author was a scientific assistant at the University of Rostock in the Research Group Knowledge Representation and at the Brandenburg Technical University of Cottbus in the Research Group Database and Information Systems of the Computer Science Institute and was supported by Prof.Dr.Bernhard Thalheim. I acknowledge my supervisor Prof.Dr.Bernhard Thalheim. The framework for the development of adaptable database design strategies is based on helpful discussions [30, 3]. The author would like to thank Ute Riebow for helping with the English.

References

- [1] M.Albrecht, E.Buchholz, A.Düsterhöft, B.Thalheim. An Informal and Efficient Approach for Obtaining Semantic Constraints using Sample Data and Natural Language Processing. In: Proc. of the Workshop on Semantics in Databases in Prague, January 13 - 16, 1995.
- [2] M. Albrecht, M. Altus, E. Buchholz, A. Düsterhöft, and B. Thalheim, The rapid application and database development (RADD) workbench - A comfortable database design tool. In: Proc. of the 7th International Conference, CAiSE 95 (eds.: J.Ivari, K.Lyytinen, M.Rossi), LNCS 932, Jyväskylä, Finland, 1995, p.327–340 <http://www.informatik.tu-cottbus.de/cs/dbis/radd/caise95/caise95.html>.
- [3] M.Altus, B.Thalheim. Strategy Support in RADD. In: Proc. of the Conference on Information Technique and Organisation - Planning, Economy and Quality, Ulm, 1995 (in German) <http://www.informatik.tu-cottbus.de/~altus/node4.html>.
- [4] M.Altus. Meta-Modelling of Conceptual Data Base Design Using a Plausibility Function. Accepted for publication at: The 6th European-Japanese Seminar on Information Modelling, May 28-31, 1996 <http://www.informatik.tu-cottbus.de/~altus/node4.html>.

- [5] M.Altus. A Modular Design Strategy for a *Flexible* Graphical Database Design Environment: An Experimental Study. In: Proc. of the 15th International Conference on Conceptual Modelling - ER'96 (Ed. B.Thalheim), in Cottbus, October 7-10, 1996 <http://www.informatik.tu-cottbus.de/~altus/node4.html>.
- [6] S.S.Anand, D.A.Bell, J.G.Hughes. EDM: A general framework for Data Mining based on Evidence Theory. In: Data & Knowledge Engineering, Volume 18, Number 3, April 1996.
- [7] P.Bachmann, W.Oberschelp, B.Thalheim, and G.Vossen. The design of RAD: Towards an interactive toolbox for database design. RWTH Aachen, Fachgruppe Informatik, Aachener Informatik-Berichte, 90-28, 1990.
- [8] A.Ballim, Y.Wilks. Beliefs, Stereotypes and Dynamic Agent Modeling. In: User Modeling and User Adapted Interaction – An International Journal (Ed.: A.Kobsa), Volume 1 – 1991.
- [9] C. Batini, S. Ceri, and S. Navathe, Conceptual database design, An entity-relationship approach. Benjamin Cummings, Redwood, 1992.
- [10] D.A.Bell, J.W.Guan, S.K.Lee. Generalized union and project operations for pooling uncertain and imprecise information. In: Data & Knowledge Engineering, Volume 18, Number 2, April 1996.
- [11] C.J.Breiteneder, T.A.Mück. Graph rewriting systems for the Entity-Relationship approach. In: Data & Knowledge Engineering, Volume 17, Number 3, December 1995.
- [12] P. P. -S. Chen, The entity-relationship model: Towards a unified view of data. ACM TODS 1, 1976, 9–36
- [13] W.B.Croft, T.J.Lucia, J.Cringean, P.Willett. Retrieving Documents by Plausible Inference: An Experimental Study. In: Information Processing & Management Vol. 25, No. 6, pp. 599-614, 1989.
- [14] J.-L. Hainaut, Catherine Tonneau, M.Joris, M.Chandelon. Transformation-Based Database Reverse Engineering. In: Proc. of the 12th International Conference on the Entity-Relationship Approach, Arlington, Texas, USA, December 15-17 (Eds.: R.A.Elmasri, V.Kouramajian, B.Thalheim), LNCS 823, Springer-Verlag, 1993, p.364–375.
- [15] T.Halpin, H.Proper. Database Schema Transformation and Optimization.. In: Proc. of the 14th International Conference on Object-Oriented and Entity-Relationship Modeling, LNCS 1021, Gold Coast, Australia, December 13-15 (Ed.: M.P.Papazoglou), Springer-Verlag, 1995, p.191–203.
- [16] M.Himsolt. Graph^{Ed} User Manual. Universität Passau, Fakultät für Informatik, Juli 1990.
- [17] InfoModeler, Evaluation Guide. Asymetrix Corporation, 1994.
- [18] R.Kass. Student Modeling in Intelligent Tutoring Systems – Implications for User Modelling. In: Kobsa, A.; Wahlster, W. (Eds.): User Models in Dialog Systems. Springer Verlag, 1989, p.233–254.
- [19] A.Kobsa. Adaptability and User Modelling in Interactive Software Systems (in German). Abstract in the German Journal of Artificial Intelligence (special issue), August, 1993.
- [20] A.J.Kok. User Modelling for Data Retrieval Applications. Vrije Universiteit Amsterdam, Faculteit Wiskunde en Informatica. 1990.
- [21] McCoy, K.F. Highlighting a User Model to Respond to Misconceptions. In: Kobsa, A.; Wahlster, W. (Eds.): User Models in Dialog Systems. Springer Verlag, 1989, p.233–254.
- [22] M.F.McTear. User modelling for adaptive computer systems: a survey of recent developments. Kluwer Academic Publishers, Artificial Intelligence Review 7, p.157–184, 1993.
- [23] O.Rauh, E.Stickel. Standard Transformations for the Normalization of ER Schemata. In: Proc. of the 7th International Conference, CAiSE 95 (eds.: J.Ivari, K.Lyytinen, M.Rossi), LNCS 932, Jyväskylä, Finland, 1995, p.313–326.

- [24] A.Rosenthal, D.Reiner. Tools and Transformations – Rigorous and Otherwise – for Practical Database Design. In: acm Transactions on Database Systems, Volume 19, Number 2.
- [25] K.Siau, Y.Wand, I.Benbasat. A Psychological Study on the Use of Relationship Concept – Some Preliminary Findings. In: Proc. of the 7th International Conference, CAiSE 95 (eds.: J.Ivari, K.Lyytinen, M.Rossi), LNCS 932, Jyväskylä, Finland, 1995, p.341–353.
- [26] V.C.Storey, C.B.Thompson, S.Ram. Understanding database design expertise. In: Data & Knowledge Engineering, Volume 16, Number 2, August 1995.
- [27] E.Szeto, V.M.Markowitz. ERDRAW 5.3: A Graphical Editor for Extended Entity-Relationship Schemata, Reference Manual. 1994.
- [28] S.L.Tanimoto. Chapter 7: Probabilistic Reasoning. In: The Elements of Artificial Intelligence Using LISP. Computer Science Press, 1987, p.239-281.
- [29] B.Thalheim. Reconsidering Key and Identification Concepts in different Database Models. Preprint CS-08-91, Computer Science Department, Rostock University, 1991.
- [30] B.Thalheim. Database Design Strategies. Computer Science Institute, Cottbus Technical University, D-03013 Cottbus, 1993.
- [31] B.Thalheim. Fundamentals of Entity-Relationship Modelling. Springer-Verlag, 1996.