The *R*apid *A*pplication and *D*atabase *D*evelopment (RADD) Workbench — A Comfortable Database Design Tool *

Meike Albrecht[2], Margita Altus[1], Edith Buchholz[2],
Antje Düsterhöft[2], Bernhard Thalheim[1]

[1] University of Rostock, Department of Computer Science
[2] Cottbus Technical University, Department of Computer Science

Abstract. In this paper we present a tool for database design which supports designers efficiently and informally to achieve correct and efficient databases.

1 Introduction

The performance of a database (especially efficiency and consistency) heavily depends on design decisions. In order to achieve an effective behaviour of the database, database designers are requested to find the best structure and the simplest basic database operations. The result of the database design process depends on the professionality of the designer and the quality of the support by a database design system. Therefore development of a comfortable database design system is an important task.

In this paper we want to present a database design system which is adaptable to a designer. It contains components which enable even novice or unskilled users the design of correct databases. An extensive support of database designers in choosing design strategies and checking correctness of design steps is contained in the approach. These components also use natural language to acquire information about databases and discuss design decisions by means of examples. Another part of the workbench transforms designed databases into equivalent and more efficient databases.

The different components in the workbench work closely together. The designer does not need to enter an information twice, all components communicate via a DataDictionary. Therefore, the designer can decide which support he/she wants to use for every design task.

In the system we use a special extension of the entity-relationship model which can be used to represent structural, semantic and behavioural information. The workbench is currently implemented in a joint project of different groups [BOT90] in Cottbus, Dresden, Münster and Rostock. We want to explain main parts of the workbench in this tool.

2 Why Another Tool Box

During the last decade several dozens of computer-aided database engineering tools (CASE) have been developed and used in practice, e.g. Accell, AD/VANCE, Aris, Axiant, Bachmann/Database Administrator, CA-ADS/Online, CA-Ideal, CASE Designer, CASE Generator, case/4/0, Colonel, COMIC, DatAid,

* This work is supported by DFG Th465/2

DataView, $(DB)^2$, DBleaf, DBMOD, DDB CASE, DDEW, EasyCASE, Enter Case, ErDesigner, ER-Win/SQL, ERWin/DBF, ERWin/ERX, Excelerator, Focus, Front & Center, GainMomentum, Gambit, Hypersript Tools, Ideo, IDEF, IEF, IEW, Informix-4GL Rapid Development System, InfoPump, Ingres/Windows4GL, Innovator, JAM, Maestro II, Magic, MTS, Natural, Ossy, PackRat, Paradigm Plus, PFXplus, Pose, PowerHouse, ProdMod-Plus, Progress 4GL, RIDL*, QBEVision, ROSI-SQL, S-Designor, SECSI, SIT, SQL Forms, StP/IM, Superbase Developer Edition, SuperNOVA, System Architect, TAXIS, TeamWork, Uniface, VCS, XPlain, and ZIM.
At present, we can distinguish three generations of database design systems ([BCN92, RoR89]).

- The first generation tools were based on the classical "waterfall" model of software development: requirement analysis, conceptual design, logical design, testing and maintenance. Most of them were platforms for a design from scratch. These tools did not support changes during the life-cycle of the database.
- Second generation tools which become available now are designed as complete workbenches for the design support over the complete life-cycle of a database. Such tools use graphic subsystems and support consistency of the design. Some of them help the user to find out which information is entered several times and/or which is inconsistent. Further, some systems generate design documentations for the complete design process. Most of the workbenches are adaptable to different platforms and can generate different translations for a given design.
- Although second generation tools are now put into practice, third generation tools are already under development. There are proposals in which manner tools can be customized. Third generation tools will be more user-friendly and user-adaptable (using, for instance, user-driven strategies which are influenced by special organizational approaches in enterprizes). Designers can use strategies which are model-dependent and have a support for reasoning in natural language. They provide a tuning support. Users can apply object-oriented development strategies.

The progress of design tools is based on software, hardware and methodology development. AI techniques contributed to design approaches appending methodological support, knowledge engineering approaches and customizability. Novel technologies like object orientation and hypertext are affecting database design systems. Hardware architectures and changing platforms support systems with better and deeper support. Databases are evolving. Therefore, design systems need to support evolution and re-engineering. New and more complex applications require better design systems. Summarizing, advanced design tools should satisfy the following requirements [BCN92]. These features are the same as for advanced CASE systems.

1. The tool needs an *advanced* and *powerful user interface*. The presentation on the screen should be consistent. Further, the user interface supports recognition of past design decisions and uses a graphical language which is simple and powerful at the same time and which is not displaying useless information.
2. *Flexibility* and *broad coverage* are important features of design systems. The tool supports the complete design process. Editors for schema and dataflow design, analyzers, synthesizers and transformers are well integrated. Different design strategies are supported. There are interfaces to external tools and different platforms. The methodology support can be adapted to the user and can be enforced in different degrees.
3. The tool set is *robust, well integrated* and has an efficient *theory support*. Design, transformation and other algorithms are complete. The tool is efficiently supporting the acquisition of semantics, not

just graphics. The performance implications of design decisions can be discussed with the designer. Alternatives can be generated. The system has the ability to cope with missing information, an aid for recovering from wrong results. Further, the tool can display different versions of the same schema and cope with bad schemas as well as good.

4. A database design tool set should be *extensible* in multiple directions.

The current development of technology enables the development of components for a third generation database design system.

But at present, there is no tool which supports a complete design methodology. Most tools currently available support only a part of the design process in a restrictive manner. There are tools which do not support integrity and/or are mainly graphical interfaces. Further, there are only very few tools which can claim to be a third generation design tool. Many design tools are restricted to certain DBMS. Since any DBMS has some implementational restrictions, a computationally efficient relational design should be different for each of the systems.

Therefore, development of a database design tool is still a big challenge.

Based on an analysis of systems supporting database design the database design system $(DB)^2$ has been developed [Tha92]. It is supporting a high-level efficient database design. The system is based on an extension of the entity-relationship model for structural design and on a graphical interface. Further, a high-level language for specification of integrity constraints and operations has been developed. Four year of extensive utilization of $(DB)^2$ in more than 100 different user groups gave us a deeper insight into the design process and the needs of database designers.

Based on the experiences with $(DB)^2$ we are developing a new design system RADD (*R*apid *A*pplication and *D*atabase *D*evelopment). The aim of this paper is to demonstrate that the requirements on database design systems can be met by current technology. First, we represent the new design system RADD. Then we present different tools of this system in more detail.

3 Overview of RADD

The RADD (*R*apid *A*pplication and *D*atabase *D*evelopment) developed in our groups does not require the user to understand the theory, the implementational restrictions and the programming problems in order to design a database scheme. 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 is extended to the *H*igher-order *E*ntity-*R*elationship *M*odel (HERM) by adding structural constructs and using integrity constraints and operations.

Basically, the system in Figure 1 consists of three major components:

- Design Tools: Since designers require different kinds of representation the design tools support graphical, procedural and logical techniques for application specification.
  - Graphical Editor: The system is based on an extended entity-relationship model which allows the user to specify graphically the structure of an application, the integrity constraints which are valid in the given application and the processes, operations and transactions which are necessary for the given application. This extension requires an easy-to-handle and advanced support for graphics.

- Customizer/Strategy support: The user interface is adapted to skills, abilities and intentions of the database designer. This tool allows customization of the user interface. It controls user steps and corrects the user interface according to designer actions.
  The designer is supported in choosing an appropriate database design strategy. Based on the chosen design strategy this tool controls and verifies design steps. Especially, completeness and consistency of a given specification is testified.

- Acquisition support: Acquisition of specifications can be supported by different strategies. This tool uses learning approaches for acquisition of structure, semantics and operations. The user interface of this tool is an example discussion.

- Natural language support: The designer who is able to express properties of his application based on natural language can be supported by moderated dialogues. During such dialogues the designer refines his/her current design. The system validates whether the specification meets certain completeness requirements. The system RADD supports German language in a specific manner. The structure and semantics of German sentences can be used for the extraction of structural, semantic, operational and behavioural specification.

- Version manager and reverse/reengineering tool: The design system stores versions of current, previous and sample specification. These specifications can be partially or completely included into current specifications or can be used for replacing parts of current specifications. This tool enables the designer to browse through given specification and to reuse already existing specifications.

– Optimization: In the tool there is a component which tries to find for a drafted database an equivalent and more efficient database.

- Behavior estimation: Based on frequency, priority and semantics of operations the complexity of the current database can be estimated in dependence of implementational techniques used by a chosen class of DBMS.

- Behavior optimization: Based on the results of behaviour estimation this tool discusses with a designer various possibilities for redesign and improvement of database behaviour. Improvement includes modification and optimization of database schemata, their corresponding integrity constraints and operations.

– Translator: This component translates the result of the design process into the language of a specific database management system that is used in the given application. This tool is developed on the University at Münster (Germany) and therefore not described in this paper.

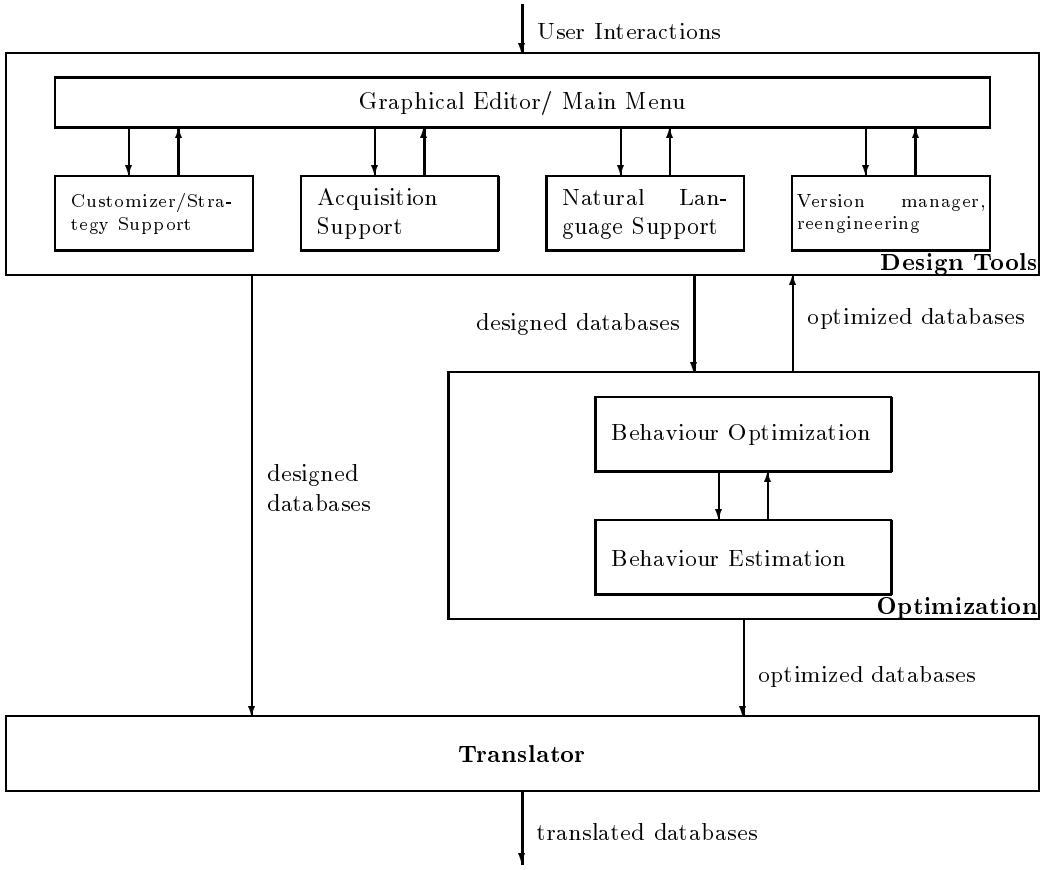

Figure 1: Overview of the RADD workbench

In the next sections we want to represent some main parts of the workbench more extensively. In section 4 we show how designers can be supported in the design steps individually. Section 5 describes a part of the acquisition support. We chose the informal and efficient acquisition if semantic constraints because this is a difficult task in database design and because semantic acquisition is NP complete. The next section demonstrates the natural language support.
Section 7 represents in which manner database drafts can be transformed into equivalent optimal databases. Therefore, behaviour estimation and behaviour optimization are demonstrated. In section 8 we give a conclusion of the paper.

4 User Guidance

There is a variety of database design strategies proposed in the literature (e.g. [Ris88, FlH89, BCN92, FuN86, Leo92]). Most approaches propose that the designer is mainly using one strategy. However, observations on database designers show that they use different strategies depending on the application properties, their skills and abilities and the chosen platform for implementation. This change in strategy

necessitates a support for database designers. Further, database designers change their strategy according to the reached stage of the scheme. Any change in strategy makes the design task more complex. Since database designers normally delay some of the design decisions, any change in strategy requires deliberate adaptation. For this reason, experienced database designers need a strategy support as well. Therefore, a user guidance tool is currently developed and included into RADD. This tool supports the designer

- in developing and applying his/her own strategy,
- in discussing decisions in designer teams,
- in tracing delayed design decisions, and
- in customizing the design workbench.

It is based on

- a framework for deriving design strategies,
- a user model, and
- a customizable graphical environment.

The Framework for Adaptable Database Design Strategies. Each design strategy needs to support a designer during a consistent development of database schemes. Thus, a strategy should be content preserving, constraint preserving, minimality preserving and update simplicity preserving [Yao85]. Analysing known design strategies, primitive steps can be extracted. These *primitives* are the basis for the strategy support in RADD. The designer can choose his/her own strategy and compose the strategy from primitives. The system RADD supports this choice based on the user model, user preferences, and characteristics of the application. On this basis, the designer can switch among bottom-up, top-down, modular, inside-out and other strategies. The *controller* derives graph grammar rules for the maintenance of consistency and for the specific support of designers. The variety of different strategies is based on the dimensions of database design which are *design directions* like the top-down or bottom-up approach or mixed strategies, *control mechanism* of design strategies and the *modularity concept*. Strategy consultancy and error control are included in the user guidance tool. Besides the primitives each design strategy is related with a set of checkpoints. The design of the unit 'borrow' in our library example involves a rough specification on the interface. The checkpoint after this step examines the interface description and the skeleton of the library example.

The Type of the User Modelling Component. The model of the user is represented *explicitly* within the design environment. The customizer makes the system *adaptable* – that is, the user can make choices among various options that effect the system's behaviour, e.g. the user may enable or disable different design strategies or restrict the output of types with respect to a certain type of cardinalities. Beyond the customizer the *adaptive* component automatically acquires knowledge about the users (designers), updates this knowledge over time, and uses the knowledge to adapt to the user's requirements. In this way it is a *dynamic* user model. Our user model contains knowledge about individual users (a set of profiles, see below) and knowledge about classes of users (e.g. users experienced in relational database design).

Figure 2: Screendump of the Graphical Editor, Strategy Support, and Customizer

The Sort of Information in our User Model. The *user* of the database design environment 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 will be supported with respect to this *classification*.

How is the User Information Acquired? The users knowledge and actions are analysed with the aim to propose the most appropriate design strategy before the scheme transformation process starts and the most likely next design step after each user design action. The user analysis is divided into two parts: the direct analysis (user's answers in an interrogation) and the indirect analysis (user's actions in the design environment, e.g. selection of types in the diagram, or selection of commands/ design primitives). In this way, the analysis contains both *explicit* and *implicit acquisition* of user model information.

Representing User Characteristics. Individual users are represented as 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 use of particular actions. A user action results in the execution of a scheme transformation applying one design primitive. Each design primitive is

described by a starting subscheme and a resulting subscheme and has a graphical representation and implementation. Therefore, action frames, the so called *user profiles*, contain small design-state-dependent patterns of the user behaviour.

When user information is needed (timing of adaptation) the system determines what are the properties of the current design state and collects profiles which agree with this. The collection of all these pieces of information is then accumulated in one large user model called the *"focus"*.

How is the Information Used? User adaptation includes the derivation of the appropriate adaptation forms (e.g. context-sensitive help, design primitive, design strategy), the dimension of adaptation (e.g. the degree of support and extent of help) and the extent and style of information the user is able to consume about adaptation as well as the realization of the user adaptation. It is a learning approach, the system learns from earlier design steps, evaluates them and implies the most likely next design step.

5 Informal Acquisition of Semantic Constraints

Correct and complete determination of semantic constraints is one of the most difficult tasks in database design because semantic constraints are formally defined and therefore designers often misunderstand them. But semantic constraints are necessary to ensure that restructuring operations can proceed correctly. Therefore a support of acquisition of correct and (as far as possible) complete sets of semantic constraints by a tool is important. Most database designers know the context of the application but they cannot express these semantic constraints. Our tool shall help those designers in specifying constraints. In this tool we use an informal approach basing on example relations. Therefore the approach is useful even for novice or unskilled designers.

Analyse of Example Relations: In the approach we first need some real-world data for the database. From these data it is derivable that some constraints cannot be valid. We want to show it by a sample relation 'Persons'.

PersonsNumber	Name		Address				Phone
	FamName	FirstName	Town	ZipCode	Street	Number	
218263	Meier	Paula	Berlin	10249	Mollstr	30	3293238
948547	Mueller	Karl	Berlin	12621	Mittelweg	281	3743654
323983	Schmidt	Anna	Rostock	18055	Gerberstr	30	8834736
239283	Weber	Peter	Rostock	18055	Gerberstr	15	9329392

The following keys cannot be valid: -0.1cm

- Adress.Number - Adress.City Adress.ZIP Adress.Street

Further, it is derivable from the instance that the following functional dependencies are not valid:

- Address.Number $\not\rightarrow\!\!\!\!\rightarrow$ PersonsNumber
- Address.Number $\not\rightarrow\!\!\!\!\rightarrow$ Name.FamName
- Address.Number $\not\rightarrow\!\!\!\!\rightarrow$ Name.FirstName
- Address.Number $\not\rightarrow\!\!\!\!\rightarrow$ Address.Town
- Address.Number $\not\rightarrow\!\!\!\!\rightarrow$ Address.ZipCode
- Address.Number $\not\rightarrow\!\!\!\!\rightarrow$ Address.Street
- Address.Number $\not\rightarrow\!\!\!\!\rightarrow$ Phone
- Address.Town Address.ZipCode Address.Street $\not\rightarrow\!\!\!\!\rightarrow$ PersonsNumber
- Address.Town Address.ZipCode Address.Street $\not\rightarrow\!\!\!\!\rightarrow$ Name.FamName
- Address.Town Address.ZipCode Address.Street $\not\rightarrow\!\!\!\!\rightarrow$ Name.FirstName
- Address.Town Address.ZipCode Address.Street $\not\rightarrow\!\!\!\!\rightarrow$ Address.Number
- Address.Town Address.ZipCode Address.Street $\not\rightarrow\!\!\!\!\rightarrow$ Phone

Invalid keys and invalid functional dependencies can be derived from one relation. Invalid exclusion dependencies between two relations and non-possible cardinalities can be found from sample relations. These invalid semantic constraints restrict search space which must be checked for semantic constraints.

Explicite Determining of Semantic Constraints: Often database designers can determine some semantic constraints, for instances a key of a relation. Therefore, explicit determination of semantic constraints is also supported in the tool, because designers who know semantic constraints do not want to search for them in examples.

Search for Further Constraints: If a designer cannot enter semantic constraints or if he cannot determine all constraints then the tool supports the search for further constraints.

All semantic constraints which are still not yet analysed (not violated by the example and not explicitly entered) could be valid and therefore they must be checked. It is possible to inquire those semantic constraints with examples. We want to show it for an inclusion dependency, only:

Persons:

Person_number	Name	First_name	City	ZIP	Street	Number
	Meier Schulz Lehmann					

Students:

Student_number	Name	First_name	Department
	Schmidt		

Is it possible that — Students.Name — contains values which are not in — Persons.Surname — (y/n) ?

Keys, functional dependencies, exclusion dependencies and cardinality constraints can also be inquired by an example discussion.

Efficient Acquisition of Unknown Semantic Constraints: The set of unknown constraints which must be checked can be very large. The number of independent functional dependencies and keys of a relation is $O(2^n)$ where n is the number of attributes of a relation. The number of unary inclusion and exclusion dependencies is $O(n^2)$ where n is the number of attributes of the whole database.

Therefore, not all unknown constraints can be checked one after the other. *Heuristic rules* are necessary which estimate the probability of the valididity of unknown constraints. These heuristics can use much vague information about the database which reflects background knowledge of the designer. Structural information, already known semantic information, sample relations and sample transactions (if they are known) are utilized in the heuristic rules.
We want to show only some heuristics.
From *attribute names* keys are sometimes derivable if the substrings -name-, -number-, -id-, -#- occur in the names. Similar attribute names can indicate existing inclusion or exclusion dependencies or foreign keys. *The same values in the instances* also point to inclusion and exclusion dependencies or foreign keys. From *transactions* specified on the databases we can conclude which attributes are probably keys, and which functional and inclusion dependencies must hold.

These heuristics try to exploit the *background knowledge of the designer* that is already implicitely contained in the database.

First, those constraints which seem to be valid are inquired with the example discussion.

There is another possibility to speed up acquisition of semantic constraints. First those candidates for semantic constraints are inquired from which most other unknown constraints are derivable.

In that way unknown semantic constraints are inquired in an informal and efficient way. The designer needs not to be able to enter formal semantic constraints, but with this tool he can determine valid constraints. These constraints are necessary in the optimization of databases.

6 The Natural Language Interface (NLI)

6.1 Motivation and Aims of the Linguistic Approach

A database designer has to use a high level of abstraction for mapping his real-world application onto an entity relationship model. In fact, most users are able to describe in their native language the entities that will form the basic elements of the prospective database, how to administer them and the processes they will have to undergo. For that reason we decided to choose the natural language German for supporting the database design process.

In this section we illustrate how natural language in a dialogue system can be used for gathering the knowledge of the designer and how it can be transfered into an extended entity-relationship model. The dialogue together with the knowledge base will be used for drawing to the designers attention special facts resulting from the syntactic, the semantic and the pragmatic analyses of the natural language input. The system makes suggestions for completing the design applying the knowledge base. We have implemented a rule-based dialogue design tool for getting a skeleton design on the basis of HERM structures.

The specification and formalisation of semantic constraints and behaviour is one of the most complex problems for the designer. Within natural language sentences the designer uses semantic and behaviour constraints intuitively. For that reason within the natural language design process we focus on extracting comprehensive information about the domain from natural language utterances. The results of the dialogue are available in the internal DataDictionary for the other tools (graphical interface, integrity checker, strategy adviser,...) of the system. Within the RADD system the designer can use these results for various forms of representation, e.g. a graphical representation. The skeleton design with the semantic constraints is also the basis for further semantic checks, e.g. of key candidates, and will restrict the search areas in the checking process.

6.2 The Structure of the NLI

The natural language interface consists of a dialogue component, a component for the computational linguistic analysis and a pragmatic component. Each component will be described in the next sections.

Dialogue Component For the acquisition of designer knowledge we decided to choose a moderated dialogue system. A moderated dialogue can be seen as a question-answer-system. The system asks for input or additional questions considering the acquisition of database design information. These questions are frames which will be updated in the dialogue process. The designer can formulate the answer in natural language sentences. Within the dialogue the results of the syntactic, semantic and pragmatic analyses will be used for controlling the dialogue.

Component for the Computational Linguistic Analysis The computational linguistic analysis consists of a syntactic and a semantic check of the natural language sentences.

The designer input into the dialogue tool is first submitted to a syntax analyser. In order to check the syntax of a sentence we have implemented an ID/LP parser (Immediate Dependence/ Linear Precedence) which belongs to the family of Unification Grammars. The parser works on the basis of a lexikon of German words and a restricted area of German sentences.

Example. 'The user borrows a book with a borrowing-slip'
one possible syntax tree:
S(NP(DET(the),N(user)),
VP(VP(V(borrows),NP(DET(a),N(book)),
PP(PRAEP(with),NP(DET(a),N(borrowing-slip)))))

Interpreting the semantics of the designer input we are using the two-step model that contains the word semantics and the semantics of the sentence. Verbs form the backbone of the sentences. We have tried to find a classification of verb semantics that can be applied to all verbs in the German language. This classification is, at this stage, independent of the domain to be analysed. To identify the meaning of sentences we have used the model of semantic roles. The units in a sentence are seen to fulfil certain functional roles corresponding to the verb classification.

Example. 'The user borrows a book with a borrowing-slip'
semantic analysis:
verbtype: verb of movement
subject: the user
object: a book
locative:? (additional question)
temporal: ? (additional question)
mode of movement: with a borrowing-slip

Pragmatic Component The aim of the pragmatic interpretation is the mapping of the natural language input onto HERM model structures using the results of the syntactic and semantic analyses. We handle this transformation as a compiler process. An attribute grammar with common rules and heuristics as semantic functions form the basis of the transformation. Common rules will be used for analysing the syntax tree structures of the syntactic analysis.

The heuristics are expressed in contextfree and contextsensitive rules and represent assumption needed for the transformation of natural language phrases into HERM structures. (E.g. The close relation of nouns and entities or the close relation of verbs and relationships.)

Example. Heuristic: Sentences with forms of the verb 'be' and an object are is-a classifications.
Rule: N(X),subject(X),V(be),N(Y),object(Y) $\rightarrow$ entity(X),entity(Y),super(X,Y).

Procedures for extracting semantic information are also started from the transformation grammar if special words are identified. Within the knowledge base these words will be marked.

Example. Heuristics: Special words which have the character of numbers will be key candidates. E.g. ISBN, street number, registration number.

Information on behaviour can be best gained from a knowledge base. Special words indicate the occurence of processes. If such words are recognized a process classification will be applied in order to capture the according post, main and preprocesses.

Example. 'The user borrows a book with a borrowing-slip.'
process analysis: borrow - synonym: lending
lending: preprocess: obtaining, registration
lending: postprocess: returning

The advantage of this strategy is the possiblity to connect not only word classes (e.g. nouns, verbs, adjectives) with HERM structures but also phrases of the sentences (e.g. genitive phase) and HERM structures.

7 Behavioural Optimization

Traditional database design is still based on waterfall approaches. The designer starts with requirement analysis, designs the conceptual scheme, normalizes this scheme and translates it into the logical scheme. Later, during physical design *tuning* of logical schemes becomes necessary. During the tuning phase operational behaviour is considered in detail. Processing requirements (planning, control, operations) are now taken into account. However, operational information is available during conceptual design. There are several reasons why processing information needs to be considered during conceptual design:

- *Efficiency* is one of the main database processing requirements. Low storage complexity and low operational complexity is an objective of database design. It can be reached using different methods. Normalization is one approach during conceptual design. Tuning is the main approach during logical and physical design. E.g. search operations on book information and update operations on 'borrow' are used more frequently than adding new readers to the library database.
- As the normalization examples in [Ull82] show, an algorithm that derives a certain normal form can generate *different normalized data schemes* from one and the same set of attributes and functional dependencies. We need to choose the scheme with the 'best' operational behaviour. In our example, any normalization which separates 'borrow' is less efficient than those which allows to treat 'borrow' as a unit.
- A 'good' conceptual scheme does not provide a good *physical realization* automatically. Additionally, for some of those physical schemes – derived with the help of statical aspects only – it can be shown that a more optimal conceptual base scheme exists which is not directly derivable from the given conceptual scheme. If this scheme is the basis of the database system then external views cannot be directly supported. E.g. the book information view should be directly derivable from the conceptual schema.
- *Integrity enforcement* aims at deriving mechanisms for simple extended operations which never violate integrity constraints in the case that the database is correct before application of the operation. For instance, removing a reader from the library requires the return of all books etc. lent by this

reader. The remove operation is a complex operation which can be derived from structural and semantical knowledge obtained during database design.
- Further, methods of *dependency protection at the physical level* are of another nature than those at the conceptual level: integrity constraints result in operational dependencies used for maintenance of the database system. E.g., simple access is supported by implementational approaches like surrogate keys, identifiers or keys like the ISBN number. The last key is not practical for the library user.

Therefore modern database design approaches advice inclusion of processing information into the conceptual design phase as well. This is possible:

- During conceptual design an important part of the knowledge which is used for tuning schemes is already available [CoG93, Sha92, Su85].
- The internal database representation is based on well-known data structuring techniques. The complexity of corresponding operational support is well-known [KoS91, Wie87].
- There exists a general mechanism [SST94, ScT94] which allows the computation of integrity maintening procedures for the database implementation.

Therefore, RADD enhances the conceptual database design phase by considering operational behaviour too. Elimination of operational bottlenecks and optimization can already be performed at the time of conceptual design. This provides a more application-oriented database development. This does *not* require *complete* logical modelling during conceptual design. Only those steps are considered which are necessary for the estimation of operational behaviour.

This approach is more complex. Thus, we have developed a tool for behavioural estimation of physical behaviour and another tool for behavioural optimization.

Behavioural Estimation: The estimizer is based on the following framework.

+ Since the trigger approach is not powerful enough we use the approach [ScT93] for the construction of consistent update operations and extended transactions. It is based on linguistic reflection. Type-safe linguistic reflection came up with the development of the ADABTPL language which has been used for deriving correct database transactions [StS90, StS91].
+ Two different cost estimation models are implemented based on characteristics of chosen DBMS.
  * The basic model is based on the conceptual information: structure, semantics, frequency and priority of operations.
  * The enhanced model is using further information on class size, index formats and on implementation strategies
+ The tool is currently extended by an expert system which is used to discuss with the designer certain bottlenecks in accordance with the chosen *application scenario*. We have included into the system different tuning rules known for some of the major DBMS. The designer introduces the main operational requirement (e.g. frequency and priority of operations and transactions). The system generates better design decisions and displays it for further discussion.

Behavioural Optimizer: Based on analysis of operational behaviour generated by the estimizer, the *behavioural optimizer* discusses with the designer various alternatives in the conceptual schema.

The tool proposes to him/her alternative schemes with estimations of operational behaviour. The designer now can choose one of the alternative schemes. If he/she chooses one of the proposed schemes the RADD

workbench computes the changed scheme on the basis of a replacement algorithm. To enable judgement of operational fitness on the conceptual level, transformation heuristics are used for derivation from the conceptual schemes.

The designer can choose further which scheme is kept for interfaces in database applications: the original scheme or the optimized scheme. But, the optimized scheme is used internally and for the translation input. If the designer chooses the original scheme then the tool generates the corresponding adapters for the treatment of the original scheme, too.

The 'Optimizer' is implemented in Standard-ML as a collection of abstract data types which are providing the representation of entity and relationship types, data dependencies, data manipulation operations and transactions. The system will be extended in order to include also more complex transactions and parametric queries.

8 Conclusion

The workbench RADD currently under development is intended to become a third generation design system. Convenient design is supported by an advanced and powerful user interface. RADD allows the specification of structure, semantics and behaviour in consistent manner. The database designer can use different design strategies and is supported even if the chosen strategy is to be changed. Since users prefer expressing their application on the basis of natural languages the design system RADD is able to extract structural, semantical and operational specification of an application from sentences stated in natural languages. The natural language interface provides an efficient support for this facility. Another important advantage of the workbench is that the database designer gets estimations on operational behaviour during conceptual database design and is supported during optimization of conceptual schemes.

References

[AlT92] M.Altus, B.Thalheim. Design by Units and its Graphical Implementation. In: Kurzfassungen des 4. GI-Workshops "Grundlagen von Datenbanken", technical report ECRC-92-13, Barsinghausen, 1992.

[Alt94] M.Altus. A User-Centered Database Design Environment. In: The Next Genaration of Case Tools, Proceedings of the fifth Workshop on NGCT, Utrecht, The Netherlands. 1994.

[BCN92] C. Batini, S. Ceri, and S. Navathe, Conceptual database design, An entity-relationship approach. Benjamin Cummings, Redwood, 1992.

[RoR89] A. Rosenthal and D. Reiner, Database design tools: Combining theory, guesswork, and user interaction. Proc. 8th ER-Conference, 1989

[BDT94] Edith Buchholz, Antje Düsterhöft, Bernhard Thalheim, Exploiting Knowledge Gained from Natural Language for EER Database Design, Technische Universität Cottbus, Reihe Informatik I-10/1994, Germany

[BOT90] 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.

[CoG93] P. Corrigan and M. Gurry, ORACLE Performance Performance. O'Reilly & Associates, Inc., 1993.

[FlH89] C.C. Fleming and B. von Halle, Handbook of relational database design. Addison-Wesley, Reading, 1989.

[FuN86] A.L. Furtado and E.J. Neuhold, Formal techniques for database design, Springer, Heidelberg, 1986.

[Kok90] A.J.Kok. User Modelling for Data Retrieval Applications. Vrije Universiteit Amsterdam, Faculteit Wiskunde en Informatica. 1990.

[Leo92] M. Leonard, Database design theory. Macmillan, Houndsmills, 1992.
[MaR 92] Heikki Mannila, Kari-Jouko Räihä, The Design of Relational Databases, Addison- Wesley 1992
[KoS91] H.F. Korth and A. Silberschatz, Database System Concepts. McGraw-Hill, 1991.
[Ris88] N. Rishe. Database Design Fundamentals. Prentice-Hall, Englewood-Cliffs, 1988.
[RoS87] L.A. Rowe and M.R. Stonebreaker. The POSTGRES Data Model. In *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, pages 83 – 96, Brighton, UK, September 1987.
[ScT93] K.-D. Schewe and B. Thalheim, Fundamental Concepts of Object Oriented Concepts. Acta Cybernetica, 11, 4, 1993, 49 – 81.
[ScT94] K.D. Schewe and B. Thalheim, Achieving Consistence in Active Databases. Proc. Ride-Ads, 1994
[Sha92] D.E. Shasha, Database Tuning – A Principle Approach. Prentice Hall, 1992.
[SiM 81] A.M. Silva, M.A. Melkanoff, A method for helping discover the dependencies of a relation, In Advance in Database Theory, eds H. Gallaire, J. Hinker, J.-M. Nicolas, Plenum Publ. 1981, S 115-133
[SST94] K.-D. Schewe, D. Stemple and B. Thalheim, Higher-level genericity in object-oriented databases. Proc. COMAD (eds. S. Chakravarthy and P. Sadanandan), Bangalore, 1994
[StG 88] Veda C. Storey, Robert C. Goldstein, Methodology for Creating User Views in Database Design, ACM Transactions on Database Systems, Sept. 1988, pp 305-338
[StS90] D. Stemple and T. Sheard, Construction and calculus of types for database systems. Advances in Database Programming Languages (eds. F. Bancilhon, P. Buneman), Addison-Wesley, 3 - 22, 1990.
[StS91] D. Stemple and T. Sheard, A recursive base for database programming primitives. Next Generation Information System Technology (eds. J.W. Schmidt, A.A. Stognij), LNCS 504, 311 - 332, 1991.
[StT94] M. Steeg and B. Thalheim. Detecting Bottlenecks & Computing better Operational Behavior on Conceptual Data Schemes, October 1994. (submitted).
[Su85] S.S. Su, Processing-Requirement Modeling and Its Application in Logical Database Design. In Principles of Database Design (ed. S.B. Yao), 1: Logical Organization, 151 -173, 1985.
[TAA94] B.Thalheim, M.Albrecht, M.Altus, E.Buchholz, A.Düsterhöft, K.-D.Schewe. The Intelligent Toolbox for Database Design RAD (in german). GI-Tagung, Kassel, Datenbankrundbrief, Ausgabe 13, Mai 1994, p.28–30.
[Tha92] B. Thalheim, The database design system $(DB)^2$. Database - 92. Proc. Third Australian Database Conference, Research and Practical Issues in Databases, (eds. B. Srinivasan, J. Zeleznikow), World Scientific, 279–292, 1992.
[Tha93] B.Thalheim. Database Design Strategies. Computer Science Institute, Cottbus Technical University, D-03013 Cottbus, 1993.
[Ull82] J. D. Ullman Principals of Database Systems. Computer Science Press, Rockville, MD, 1982.
[Wie87] G. Wiederhold, File Organization for Database Design. McGraw-Hill, 1987.
[Yao85] S.B. Yao (ed.) Principles of database design, Volume I: Logical organizations. Prentice-Hall, 1985.

This article was processed using the LaTeX macro package with LLNCS style