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[2]

[1] University of Rostock, Department of Computer Science
[2] Brandenburgian Technical University of Cottbus, Institute of Computer Science
meike | buch | duest @ informatik.uni-rostock.de
altus | thalheim @ informatik.tu-cottbus.de

Abstract. We present a workbench 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 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.

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, the *H*igher-order *E*ntity-*R*elationship *M*odel (HERM), which is used for representing structural, semantic and behavioural information. The workbench is currently implemented in a joint project of different groups [BOT90] in Cottbus, Dresden, Münster and Rostock. In the following sections we explain main parts of the workbench.

* This work is supported by DFG Th465/2.

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. [1]). 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. 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].

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.

[1] Accell, AD/VANCE, Aris, Axiant, Bachmann/Database Administrator, CA-ADS/Online, CA-Ideal, CASE Designer, CASE Generator, case/4/0, Colonel, COMIC, DatAid, DataView, $(DB)^2$, DBleaf, DBMOD, DDB CASE, DDEW, EasyCASE, Enter Case, ErDesigner, ERWin/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.

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*. 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.
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. Five years 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 an Overview of the new design system RADD. Then we illustrate different tools of this system in more detail.

3 Overview of RADD

The RADD (*R*apid *A*pplication and *D*atabase *D*evelopment) 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 the extended entity-relationship model (HERM) whereby structural constructs are added and integrity constraints and operations are used.
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. Further, the editor enables the designer to represent the complete design information. The designer develops the structure, static and dynamic semantics, and the operations of the application. Several components support the specific design tasks, e.g. semantics can be developed using an abstract approach, using an approach based on examples or refining the meaning of natural language sentences.
  - 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. The designer is supported in choosing an appropriate database design strategy. Based on the chosen design strategy this tool controls and verifies design steps.
  - 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 are used for the extraction of structural, semantical, operational and behavioural specification.
  - Version manager and reverse/reengineering tool: The design system stores versions of current, previous and sample specifications. These specifications can be partially or completely included into current specifications or can be used for replacing parts of current 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. It embodies gates to different (relational, object-oriented, hierarchical and Network) DBMS and can be used also as an input for the design process.

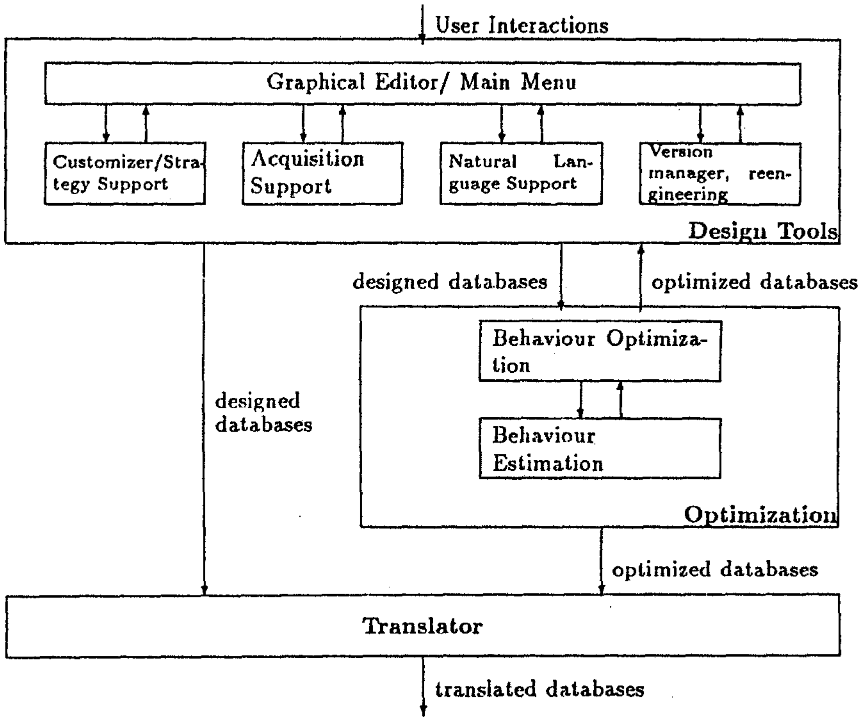

Figure 1: Overview of the RADD workbench

In the next sections we present some main parts of the workbench in detail. In *section 4* we show how designers can be supported in the design steps individually. *Section 5* describes the semantic acquisition as a part of the acquisition support. In *section 6* we demonstrate the natural language support. *Section 7* discusses the utilization of the workbench. In *section 8* we give a conclusion of the paper and consider future work.

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* [Tha94]. 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 User Modelling Component. The strategy support is based on a model of the user. The behavior of a user depends on this model, on the application area and on the team support. The user model is *explicitly* represented by a set of *profiles* in the workbench. For instance, designers experienced in entity-relationship models develop their applications differently than those designers which are used to relational database design. The customizer uses this information and generates an optimal environment for the designer. This *adaptable* environment can be enabled or disabled. Further, the RADD component guard the actions of users. If the profile of the user changes then the system generates a proposal for a change of the RADD environment. This *dynamic* user model also represents changes in knowledge, skills and capabilities.

The Information in our User Model. The *user* of the database design environment is classified regarding his properties, his capabilities, his preferences (kinds of

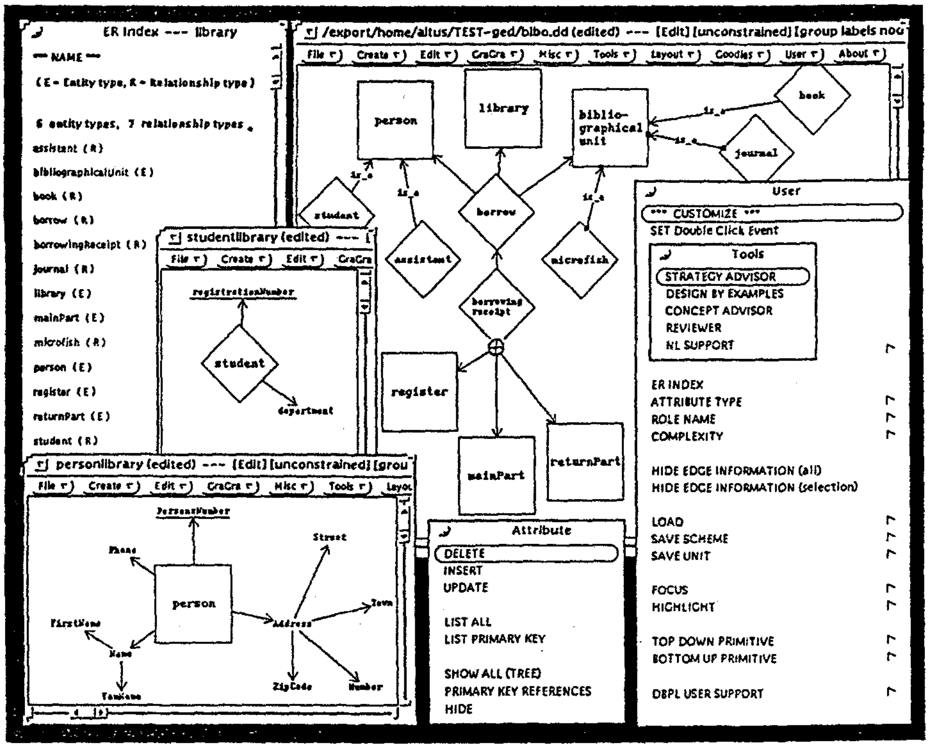

Figure 2: Screendump of the Graphical Editor

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*.

Acquisition of the User Information. 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.

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 use of particular actions. A user action results in the execution of a scheme transformation which is an application of one specific 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 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"*.

Utilization of the Information. User customization 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), the extent and style of information which enables the user to use this adaptation, and the realization of the user adaptation. Since this approach is similar to learning systems, the system accumulates earlier design steps, evaluates these steps and derives 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 knowledge about the application in semantic constraints. Our tool shall help designers in specifying constraints by using an informal approach basing on example relations.

Analysis 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:

- 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 $\longrightarrow$ PersonsNumber
- Address.Number $\longrightarrow$ Name.FamName
- Address.Number $\longrightarrow$ Name.FirstName
- Address.Number $\longrightarrow$ Address.Town
- Address.Number $\longrightarrow$ Address.ZipCode
- Address.Number $\longrightarrow$ Address.Street
- Address.Number $\longrightarrow$ Phone
- Address.Town Address.ZipCode Address.Street $\longrightarrow$ PersonsNumber
- Address.Town Address.ZipCode Address.Street $\longrightarrow$ Name.FamName
- Address.Town Address.ZipCode Address.Street $\longrightarrow$ Name.FirstName
- Address.Town Address.ZipCode Address.Street $\longrightarrow$ Address.Number
- Address.Town Address.ZipCode Address.Street $\longrightarrow$ 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.

Explicit Determining Semantic Constraints. Often database designers can determine some semantic constraints, for instance 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/she 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:

PersonsNumber	FamName	FirstName	City	ZIP	Street	Number
	Meier					
	Schulz					
	Lehmann					

Students:

StudentsNumber	FamName	FirstName	Department
	Schmidt		

Is it possible that — Students.FamName — contains values which are not in — Persons.FamName — (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 can reach $O(2^n)$ where n is the number of attributes of a relation. The maximal 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 plausibility of the validity of unknown constraints. These heuristics can use much vague information about the database which reflects *background knowledge of the designer* that is already implicitely contained in the database. 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* sometimes keys are 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.
First, those constraints which seem to be valid are inquired with the example discussion.
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 by an example discussion, only. These semantic constraints are necessary for the translation and optimization of databases.

6 The Natural Language Interface (NLI)

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.

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. The NLI is described in more detail in [BDT94].

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.

Moderated Dialogue. 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.

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.
- 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.

Pragmatic Interpretation. 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. The advantage of this strategy is the possibility to connect not only word classes (e.g. nouns, verbs,

adgjectives) with HERM structures but also phrases of the sentence (e.g. genitive phrase) and HERM structures.
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. The acquisition of the semantic information will be instantiated using the attribute grammar within the process of capturing structural information.
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.

7 Using the Workbench

Practical Applications. The workbench is supporting large and complex design tasks and assists in team work. Currently, RADD is used in two large projects in Cottbus Technical University.

- The environmental system project intends to evaluate the potentiality of the southern Brandenburgian landscape after and during extensive coal mining. It is the largest ongoing project at Cottbus Tech. The size of the databases to be used in an integrated manner is estimated by tens of TB. The databases represent the industrial, mining, geological, hydrological, biological, chemical and (fertile) soil information of the area. They are used for simulation of different development scenarios. Re-engineering existing databases is a part of this project. The environmental system integrates existing soil, geographic, geologic and hydrographical DBMS. Sometimes, the documentation of the systems is missing. The user profiles of developers of these databases are diverse. Users of RADD are all kinds of environmental engineers, chemist and geologists. Most of them do not have a database background.
- The Campus Information System will replace the management information system. It includes library, mail and other services on a www basis. RADD is used for the development of databases and workflow management.

Further, a number of smaller student projects uses RADD as well.
A demo of RADD can be found in a public telnet directory of our institutes. RADD will be demonstrated during the German computer exhibition CeBit 95.

Generating Design Strategies, User Support and Consistent Database Design. A typical application of RADD would be the following. After modeling of users, project groups and application areas the system supports mixed, modular design of a complex application by a team. The natural language interface is used for the development and refinement of a skeleton of the application which is the basis for modular design and for the distribution of design tasks among the team members. Using their own design strategies the team members model their part of the application in a toggle mode (consistent structural design, semantics acquisition, design of operations and transactions, and evaluation of operational bottlenecks). Since the graph grammar supports only step-by-step transformations misconceptions can be detected early. Checkpoints are used to enforce completion of design steps before other design steps can be initiated. At the end of the design, the transformation of the complete design into the logical and physical language of the chosen implementation platform is performed.

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. The optimizer computes operational bottlenecks in the design and proposes partial changes in the design.
Future work will concentrate on repository support and complete systems design. Further RADD will support object-oriented DBMS and open integrated systems. In parallel we develop an approach for reusing database designs and for integrating existing designs into other applications.

References

[ABDT95] M.Albrecht, E.Buchholz, A. Düsterhöft, B.Thalheim: An Informal and Efficient Approach for Obtaining Semantic Constraints using Sample Data and Natural Language. Workshop "Semantics in Databases", 1 - 11, Prague, 1995.

[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.

[BDT94] Edith Buchholz, Antje Düsterhöft, Bernhard Thalheim, Exploiting Knowledge Gained from Natural Language for EER Database Design. Technical Report I - 10 - 1994, Computer Science Institute, Cottbus Technical University, 1994.

[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.

[KoS91] H.F. Korth and A. Silberschatz, Database System Concepts. McGraw-Hill, 1991.

[Leo92] M. Leonard, Database design theory. Macmillan, Houndsmills, 1992.

[MaR92] Heikki Mannila, Kari-Jouko Räihä, The Design of Relational Databases. Addison-Wesley, 1992.

[Ris88] N. Rishe, Database Design Fundamentals. Prentice-Hall, Englewood-Cliffs, 1988.

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

[RoS87] L.A. Rowe and M.R. Stonebreaker. The POSTGRES Data Model. In Proceedings of the Thirteenth International Conference on Very Large Data Bases, 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 and 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.

[Tha94] B.Thalheim. Database Design Strategies. Technical Report I - 02 - 94, Computer Science Institute, Cottbus Technical University, 1994.

[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.