In H. Weber, editor, Proc. 1st Int. Conf. on System Development Environments and Factories, London, 1990. Pitman.

CADDY : Computer-Aided Design of Non-Standard Databases

G. Engels, U. Hohenstein, K. Hülsmann*, P. Löhr-Richter, H.-D. Ehrich
TU Braunschweig, Informatik, Abt. Datenbanken, Postfach 33 29, D-3300 Braunschweig

Abstract

The CADDY project is concerned with the development of an integrated environment for Computer-Aided conceptual Design of non-standard Databases. The environment comprises tools for specification, analysis, and automatic prototyping of conceptual database schemas. In the CADDY project, a conceptual schema consists of a four layered description formed by the data, object, evolution, and action layer. All these layers have a well-defined formal semantics that forms the basis for analysis and automatic prototyping.
The system is currently being implemented on SUN workstations in the programming language C. It uses the X-Window system for realizing the user interface and the INGRES database system for installing prototyping databases.
At the current state, syntax oriented editors for the data and object layer are completed. Syntax oriented editors for the evolution and action layer as well as prototyping tools are currently under development.

1 Introduction

Most complex software systems have a common characteristic in that they administer a lot of structured data. This means that they provide a fast access to a single datum during the runtime of the software system and that they use a database system for permanent storage of the data. Examples of such software systems are office automation systems or software development environments, where a lot of structured documents are manipulated and stored in a central database. Traditional database systems are not appropriate for the realization of such database application software, since they do not provide adequate means for application-oriented modelling and handling of these complex structured data. Therefore, non-standard databases have to be designed and realized as central components in such complex software systems. In this sense, the design of non-standard databases is one important task during the software development process. Like all other software development activities, this task can be supported by appropriate tools in a database design environment. It is the goal of project CADDY to design and implement such a database design

---

K. Hülsmann's work is supported by Deutsche Forschungsgemeinschaft (Az.: En 184/1-1)

environment, which is concerned with the entire design process of non-standard databases; in particular the **conceptual design phase** of the database life cycle. The main components of the database design environment CADDY are formed by

- a **group of editors** to facilitate the input and modification of a conceptual database schema, i.e., to support the specification of the static structure and dynamic behaviour of a non-standard database,
- **tools to analyze and to check** the consistency of the conceptual database schema, and
- **tools used for means of prototyping**, i.e., for installing a prototype database on an existing database system, filling it with automatically generated test data, and executing actions on that prototype database.

As a basis for these tools, we have a conceptual data model (/HNSE 87/) provided with a well-defined, mathematical semantics (/HoGo 88/). According to this data model, the specification of a conceptual database schema consists of **four layers**: (1) the data layer for the specification of application-dependent attribute domains, (2) the object layer, based on an extended Entity-Relationship model, to specify the static structure of the database, (3) the evolution layer to describe permitted state sequences of the database, and (4) the action layer to specify permitted actions on the database (/EHNSE 88/).

There are other groups working in similar directions, but concentrating on different aspects of database design (/ADD 85/, /BDRZ 84/, /OSLS 86/). Some of them employ different semantic data models like SHM+ (/BR 84/), BIER (/EKTW 86/), or TAXIS (/MW 80/). Most groups restrict their efforts to modelling the object layer. The distinction between the object and the data layer is not made by any other group; however, the distinction seems essential to us. Only few approaches integrate behavioural aspects (actions) into conceptual design. Generally, their behaviour specifications are based on Petri-net systems (/ADD 85/, /EKTW 86/, /OSLS 86/). Dynamic integrity constraints are only covered as far as they are expressible in terms of action specifications. Among the project groups developing design tools, only a small number pursue the realization of a highly integrated design environment (/ADD 85/, /Ja 88/). In comparison with these groups, we use a semantically well-defined data model and handle both static and, at the evolution layer, dynamic integrity constraints. Furthermore, the database design environment CADDY is planned to be a highly integrated system, which is expected to result in a uniform user interface, and thereby increase the acceptance of the environment by a database designer.

2 Functionality

In this section, we illustrate our 4-layered specification approach by a more detailed description of the functionality of the design environment CADDY. Before describing the tools in detail, we postulate some common characteristics:

- At first, all tools are combined to an **integrated environment**. This results in a uniform user interface. The structure of screen displays, the dialogue of the user with the tools, and the command languages are homogeneous and compatible.

- As a consequence, the tools support **incrementality**. This enables the user to specify the schema stepwise and to refine or correct it later. At any time, the user can activate any tool which makes sense in the current situation.
- The tools are **command driven** and **syntax oriented** to relieve the user from syntactical burden. This means that the user can activate only those editing commands which do not violate the context free syntax of the currently handled document. Furthermore, any context sensitive errors are detected and indicated immediately after the input of an editing command.
- Finally, the environment is **workstation oriented**. This provides, among others, the use of a window system for the user dialogue.

Figure 1 gives an impression of how these characteristics are realized at the user interface of the environment. It shows a snapshot of a design session, where different tools have been activated by the user and tool specific information has been displayed in corresponding, possibly overlapping windows.

All tools of the environment CADDY can logically be subdivided into three classes, namely syntax oriented editors, analyzing tools, and tools for rapid prototyping.

The **syntax oriented editors** are used to design a database schema. Consequently, they follow the 4-layered specification approach:

(1) A text editor is offered for the data layer. This editor allows the specification of arbitrary data types. In addition to standard types like INTEGER or TEXT, the user can define more structured, application-oriented data types. Figure 1 shows a special window called *'Data Type View'*, which contains the specification of the data type POINT including an operation 'distance' (of two points). We propose an algebraic specification approach of data types by equations (/EM 85/). In the case of figure 1, the equation that describes the effect of 'distance' has to be completed with the text editor to obtain a correct specification of the data type POINT.

(2) The data types are used for attribute domains at the object layer. Here, object types together with their attributes and the relationships are specified using an extended Entity-Relationship (EER) model (/HNSE 87/, /HoGo 88/). We extend the classical ER model (/Ch 76/) by the concepts of type construction (specialization/generalization (/SS 77/), partition), object-valued attributes (components), an extended key-mechanism, and derived information. The design of an EER schema is supported by a graphical editor using a graphical representation of the schema called EER diagram. Figure 1 shows a further window *'EER Diagram View'* containing a preliminary design of special kinds of waters. The triangle denotes a partition of WATERS into LAKE and RIVER both inheriting the attributes from WATERS implicitly. The ovals represent attributes with possibly user-defined data types as domains. Key attributes are given a broad dot. The relationship 'flows-into' is constrained by two cardinalities. Thus, each river must flow into exactly one water (1,1), but there may be none or several rivers that flow into the same water (0,*). Hexagons represent integrity constraints and derivation rules, which are displayed in a further window, named *'Constraints View'*.

(3) At the evolution layer, all admissible database state sequences are specified by dynamic integrity constraints. Both static and dynamic constraints are expressed by formulae of an

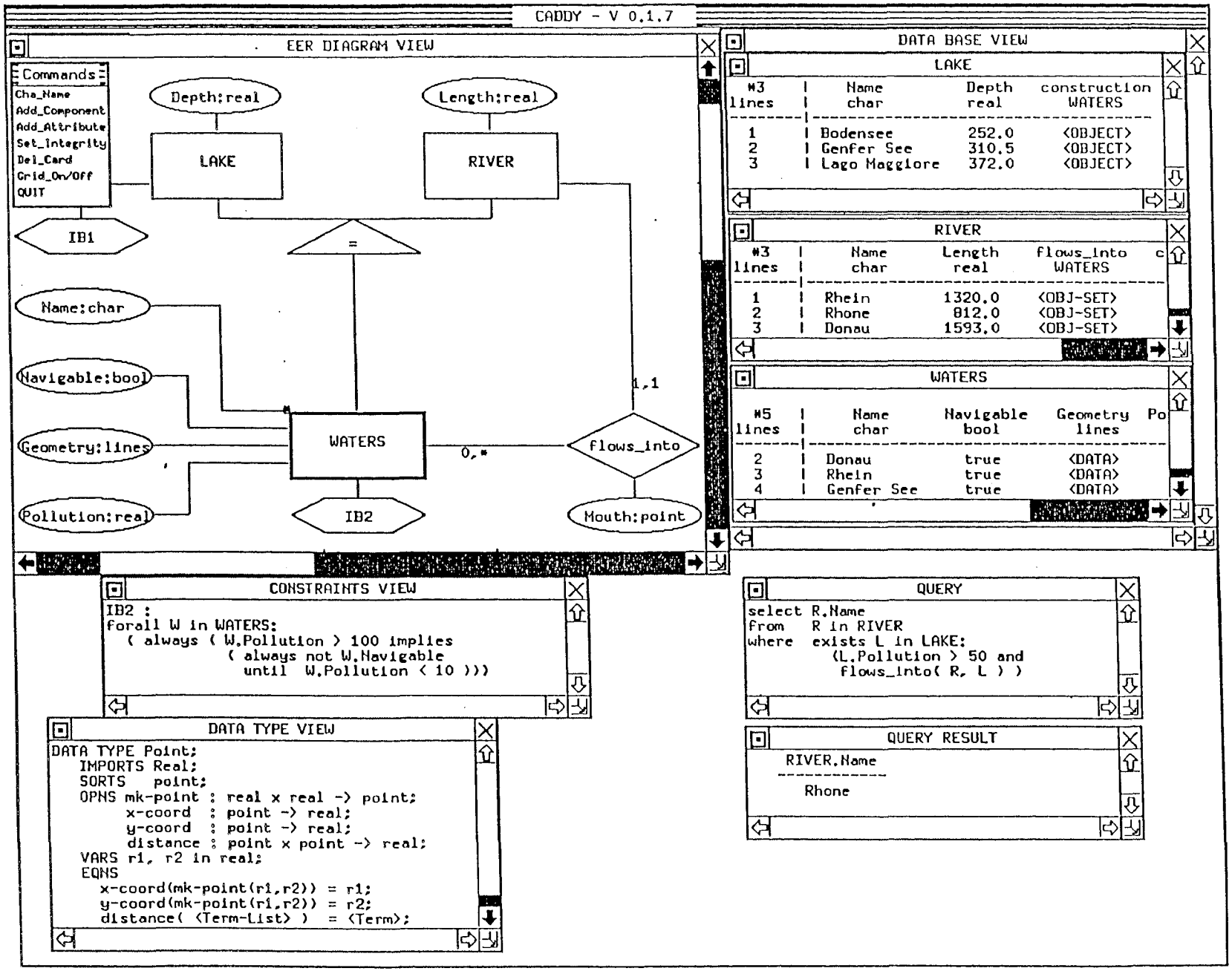

Figure 1: Snapshot of the screen layout during a design session

EER calculus (/HoGo 88/), extended by temporal quantifiers (/No 89/). Again, a corresponding text editor is given. Figure 1 contains a *'Constraints View'* with a dynamic integrity constraint corresponding to the hexagon 'IB 1' in the *'EER Diagram View'*. This dynamic integrity constraint requires that, if the pollution rate of a water is higher than 100, this water remains unnavigable until the rate is less than 10.

(4) Actions, which alter the database, are specified in the action layer. At the moment, we provide a set of so-called elementary actions, which are automatically derived from the description of the structure of the database and satisfy all model inherent constraints per definition. For instance, we have an elementary action 'insert-RIVER'. An activation of this action inserts the river, the attributes of which are given as parameters. As side effects, it causes the insertion of the corresponding water (the river "really is") and the establishment of the relationship 'flows-into' since every river has to flow into a water. Consequently, more parameters are required than the ones of the inserted river. Elementary actions can be used to compose arbitrary actions using procedural control structures like sequence, alternative, and loop. Again, a special syntax oriented text editor is given to support the user.

Although these four layers suggest a certain specification methodology in the order of the layers, an **iterative and incremental** work is possible. For example, in designing the EER diagram of figure 1, it is desirable and possible to turn back to the data layer for defining data types further needed as attribute domains.

Besides these tools for editing a conceptual database schema, there are **tools** to **analyze** the specified schema. The most important one helps the user to check whether the database behaviour specified in the evolution and the action layer is consistent and coincides with the user's intention. This is done by the help of specific (state) transition graphs which are automatically constructed from the dynamic integrity constraints of the evolution layer (/LS 87/).

In designing a database schema, it is helpful and informative for the user to execute the specified actions as early as possible. Thus, we provide tools for **rapid prototyping**. These tools can be divided into three parts:

1. At first, a prototype database is installed according to the specified structure, and afterwards automatically filled with artificial data produced by a test data generator.
2. This prototype database can now be used to execute update actions or queries on it. Queries are formulated in a powerful SQL-like query language which is based on the EER calculus proposed in /HoGo 88/. Such queries are specified in the *'Query'* window (cf. figure 1), and the result is displayed in the *'Query Result'* window. Update actions are executed interactively. The user is asked by the system for all needed actual parameter values. To enable an execution of operations on user-defined data type values, the designer has to extend the algebraic specification of the data types by an implementation in a simple programming language. This language offers some standard type constructors like list, set, or record, and the usual control structures. During the execution of update actions, an integrity monitor controls whether any static or dynamic integrity constraint is violated. If a violation occurs, the action is rejected. Therefore, a consistent database state can always be guaranteed.

3. A special database browser is provided to show the user the current database state. This browser allows structure oriented navigation in the database. For instance, the *'Database View'* in figure 1 displays the information about all waters currently present in WATERS, RIVER, and LAKE. While atomic attribute values like Name or Length are displayed immediately, component values or relationships are represented by placeholders. By selecting such a placeholder, all attribute values of the corresponding object(s) are displayed. Furthermore, the database browser has an editing facility which enables the user to change the database by substituting the displayed data.

3 Realization

Analogously to the above mentioned external characteristics of the functionality of the environment, there are some common, internal characteristics for the realization of the tools:

- All tools use a common **project database** to communicate.
- All documents, manipulated by the tools, are internally represented by a uniform data structure, the so-called **abstract syntax graph** (/En 86/).
- The use of **basic components**, like a window system on the basis of X-windows or the project database, guarantee a high degree of portability.

Let us now explain the currently running realization of the tools in more detail.

An efficient realization of the syntax oriented **schema editors** enforce an internal representation of the currently handled schema part, which expresses the context free structure explicitly. Furthermore, the detection of any context sensitive error is much easier, if the internal data structure expresses context sensitive interrelations in the document explicitly, as well. Therefore, we have chosen abstract syntax graphs as internal representation of all documents (/En 86/). This kind of an internal data structure was originally developed for the realization of the software development environment IPSEN (/Na 85/). Such abstract syntax graphs consist of abstract syntax trees to represent the context free structure, and edges to express the context sensitive interrelations. Any further non-structural information is kept in additional node attributes. For instance, the syntax graph for the representation of EER-diagrams contains node attributes to store the geometrical position of the corresponding symbols in the diagram. Using such an abstract representation of a document, each editing activity consists of three steps:

(a) The user selects a command (for instance, by a mouse click in an offered command menu),
(b) this command causes a modification of the underlying syntax graph, and
(c) a so-called unparsing or pretty-printing process generates and displays the modified external representation in a corresponding window on the screen.

The realization of the main part of the **analyzing tools** consists of an algorithm for the construction of so-called transition graphs (/LS 87/). Transition graphs are a special kind of finite automata accepting state sequences of objects in the data base, which satisfy the corresponding dynamic integrity constraint. For instance, the transition graph for the dynamic integrity constraint IB 1 of the schema definition in figure 1 consists of three state nodes.

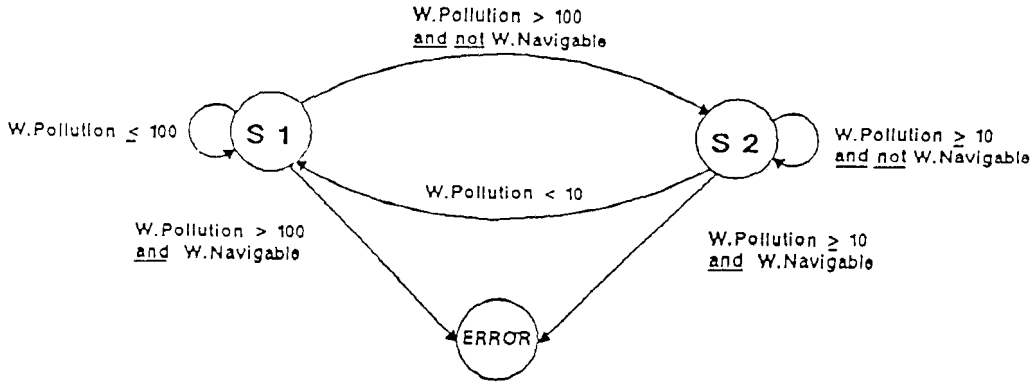

Figure 2: Transition graph for IB 1 of figure 1

Each object of type WATERS fulfills one of these three states at each point in time during the object's lifetime. At the beginning, each object fulfills state S1. If the pollution exceeds the value 100 and the object is not navigable, this object moves in state S2 and remains there until the pollution is lower than 10. If the dynamic integrity constraint is violated, the object switches to the error state. These transition graphs are used for monitoring dynamic integrity constraints (see below). During the static analysis of the specified schema, the construction of transition graphs indicates to the user whether a dynamic integrity constraint is unsatisfiable. This arises, if the constructed transition graph is empty or only consists of error nodes.

Since there does not exist any database management system for the EER-model (up to this stage), the specified database schema has to be interpreted in some way to realize the **prototyping** facility. In our approach, data layer and object layer of the conceptual schema get transformed into a relational database schema (/HNS 86/). For instance, the object type RIVER is represented by the following relation

```
R#RIVER ( RIVER#  : int;
          Length  : real;
          IS#     : int    );
```

Each relation includes a system-defined (surrogate) key, e.g. RIVER#, to identify the objects of type RIVER. The object type WATERS is partitioned into the types RIVER and LAKE. This is expressed in the relation by the additional attribute IS# corresponding to the surrogate key WATERS# of WATERS. Then, a prototype database is installed on an existing relational database system. Afterwards, any invoked query or update action is transformed into (sequences of) SQL-operations and is executed on the relational prototype database. Subsequently, query results have to be retransformed to the conceptual level, before they are shown to the user. During the execution of update actions, all explicit integrity constraints are supervised by an integrity monitor. Here, the above mentioned transition graphs are used to check whether integrity constraints are violated (/Hü 88/, /LS 87/).

All these tools for editing, analyzing, and executing a database schema are integrated in the design environment CADDY. Figure 3 sketches the software architecture of the environment. The whole system is based on three components: the input-/output-system (I/O-system), the project

database and the prototyping database. The **I/O-system** ensures a uniform interface of all tools to the user of CADDY. The main part of this component is a window management system, which encapsulates special window types like menu, text, or graphics windows. The window management system is realized by use of X-Windows (/SG 86/). This guarantees a high degree of portability of the whole environment. The central **project database** keeps all information used by different tools. All this information is internally represented as abstract syntax graphs. So we have, for instance, syntax graphs to represent the different parts of a schema, to represent transition graphs, or to represent the relational database schema. Currently, this project database is realized by an efficient data structure in the main memory, which bases on the implementation of a so-called graph laboratory (/Eb 87/). The **prototyping database** contains the testing data. It is a relational database in a commercial relational database system with an SQL-interface.

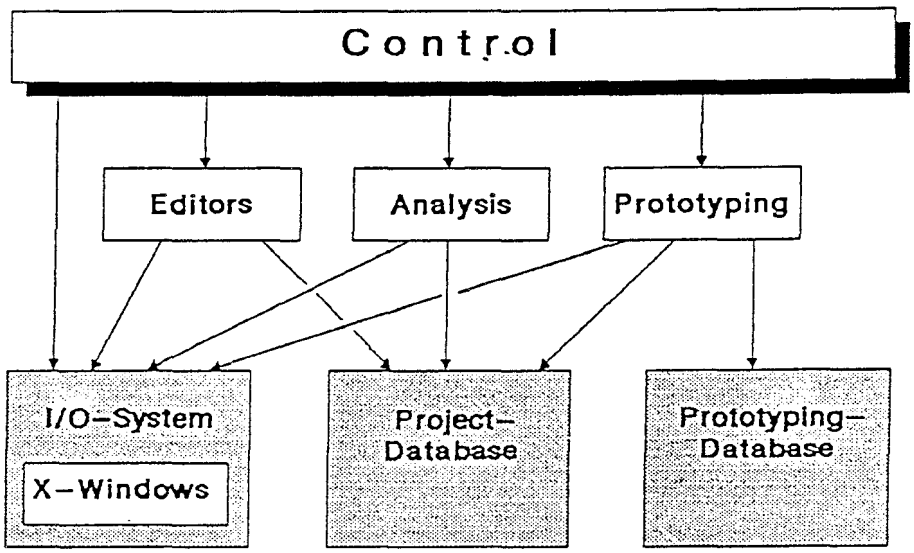

Figure 3: Rough sketch of the software architecture of CADDY

4 Conclusions

The realization of the first prototype of CADDY is currently under development. Based on earlier experiences with the implementation of individual tools, a new design phase has started in the beginning of 1988. In summer 1988, the implementation of parts of the intended tool set has begun in the programming language C on a 68020-workstation (by Integrated Solutions). Up to now, the implementation of the I/O-system and prototype versions of the syntax-aided text editor for abstract data type specification (/Os 88/) and of the graphical editor for EER-diagrams has been completed. These parts are being ported currently to a SUN 3/60 workstation. Other tools are being investigated and implemented by several diploma theses, for instance, the transformation of the schema into a relational schema, the database browser, the construction of transition graphs, the language and corresponding editors for dynamic integrity constraints, queries and update actions, or the integrity monitor.

Acknowledgements

We gratefully acknowledge the engaged work of Chr. Beer (EER diagram editor), of H. Heß and G. Schröder (I/O-system), of our colleague P. Herr, and of all the students working hard for the progress of CADDY. Also, we are indebted to Prof. Ebert (EWH Koblenz) for his permission to use the graph laboratory developed by his group.

References

/ADD 85/ Albano, A./ DeAntonellis, V./ DiLeva, A. (eds.), *Computer-Aided Database Design: The DATAID project.* North-Holland, Amsterdam 1985

/BDRZ 84/ Brägger, R./ Dudler, A./ Rebsamen, J./ Zehnder, C.A.: *Gambit - An Interactive Database Design Tool for Structures, Integrity Constraints, and Transactions.* Proc. IEEE Int. Conf. Software Engineering, Los Angeles, April 1984, **399 - 407**

/BR 84/ Brodie, M.L. / Ridjanovic, D.: *On the Design and Specification of Database Transactions.* In Brodie / Mylopoulos / Schmidt (eds.) : On Conceptual Modelling, New York Springer 1984

/Ch 76/ Chen, P.P.: *The Entity-Relationship Model: Towards a Unified View of Data.* ACM Transactions on Database Systems, Vol. 1, No. 1, 1976

/Eb87/ Ebert, J.: *A Versatile Data Structure for Edge-Oriented Graph Algorithms.* CACM, Vol. 30, No. 6, 1987, **513 - 519**

/EHNSE 88/ Engels, G./ Hohenstein, U./ Neugebauer, L./ Saake, G./ Ehrich, H.-D.: *Conception of an Integrated Database Design Environment (in German).* In F. Oertly (ed.): Proc. DBTA/SI Data Dictionaries und Entwicklungswerkzeuge für Datenbank-Anwendungen, ETH Zürich, March 1988, Zürich: Verlag der Fachvereine an den Schweiz. Hochschulen und Techniken 1988

/EKTW86/ Eder, J./ Kappel, G./ Tjoa, A M./ Wagner R.R.: *BIER: The Behaviour Integrated Entity Relationship Approach.* In S.Spaccapietra (ed.): *Entity-Relationship Approach: Ten Years of Experience in Information Modelling,* Proc. of the 5th Int. Conf. on Entity-Relationship Approach, Dijon, France, November 1986

/EM 85/ Ehrig, H./ Mahr, B.: *Fundamentals of Algebraic Specification 1.* Berlin: Springer 1985

/En 86/ Engels, G.: *Graphs as Central Data Structures in a Software Development Environment (in German).* VDI-Fortschritt-Berichte Nr. 62, Düsseldorf: VDI-Verlag 1986

/ER 88/ C. Batini (ed.): *Proc. of the 7th Int. Conf. on Entity-Relationship Approach.* Rome (Italy) 1988

/HoGo 88/ Hohenstein, U./ Gogolla, M.: *A Calculus for an Extended Entity-Relationship Model Incorporating Arbitrary Data Operations and Aggregate Functions.* In /ER 88/

/HNS 86/ Hohenstein, U./ Neugebauer, L./ Saake, G.: *An Extended Entity-Relationship Model for Non-Standard Databases.* Proc. Workshop "Relationale Datenbanken" in Lessach, Report Nr. 3-86, Comp. Science Dept. , TU Clausthal-Zellerfeld 1986

/HNSE 87/ Hohenstein, U./ Neugebauer, L./ Saake, G./ Ehrich, H.-D.: *Three-Level Specification of Databases Using an Extended Entity-Relationship Model.* Proc. GI-Fachtagung "Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen", R.R.Wagner, R.Traunmüller, H.C.Mayr (eds.), Informatik-Fachbericht Bd. 143, Springer-Verlag, Berlin 1987, **58-88**

/Hü 88/ Hülsmann, K.: *Design of a System for Monitoring Dynamic Integrity Constraints (in German).* Diploma Thesis, Comp. Science Dept., TU Braunschweig, January 1988

/Ja 88/ Jarke, M.: *The DAIDA Environment for Knowledge-Based Information Systems Development.* Report MIP 8813, University of Passau, 1988

/LS 87/ Lipeck, U.W. / Saake, G.: *Monitoring Dynamic Integrity Constraints Based on Temporal Logic.* Information Systems, Vol. 12 (1987), **255-269**

/MW 80/ Mylopoulos, J. / Wong, H.K.T.: *Some Features of the TAXIS Model.* Proc. of the 6th VLDB Conference, Montreal 1980, **339 - 410**

/Na 85/ Nagl, M.: *An Incremental Programming Support Environment.* In Computer Physics Communications 38, Amsterdam: North-Holland, **245 - 276,**

/No 89/ Nordwig, P.: *Design of a Language for the Formulation of Dynamic Integrity Constraints in an Extended Entity-Relationship Database Schema (in German).* Diploma Thesis, TU Braunschweig 1989

/Os 88/ Osterhold, A.: *Design and Implementation of a Syntax-Aided Editor for the Manipulation of Data Type Specifications (in German).* Diploma Thesis, TU Braunschweig 1988

/OSLS 86/ Oberweis, A./ Schönthaler, F./ Lausen, G./ Stucky, W.: *Net based conceptual modelling and rapid prototyping with INCOME.* In: Proc. of the 3rd Conference on Software Engineering, AFCET, Paris 1986, **165 - 176**

/SG 86/ Scheifler, R.W./ Gettys. J.: *The X Window System,* ACM Transactions on Graphics, Vol. 63, 1986

/SS 77/ Smith, J.M./ Smith, D.C.P.: *Database Abstractions: Aggregation and Generalization.* ACM Transactions on Database Systems Vol. 2, 1977, **105-133**