Reuse of Database Design Decisions

Meike Klettke
University of Rostock
Database Research Group

Abstract

Reuse of available databases can support database design and reverse-engineering of databases by allowing design decisions to be derived from existing databases.

This article proposes a method for reusing databases similar to the approach used in case-based reasoning. Similar databases, or similar parts of databases, are first determined. We then discuss the information to be reused and how it can be validated. Two methods for building libraries are suggested for use in this process.

1 Motivation

Database design is the process of determining the structure of a database, semantics and its behavioral specifications. For this process, a designer can only use informal descriptions about the application, making database design quite difficult and time-consuming, and the results often depend on the designer creativity and skill. However, the design process is crucial because the usability of a database depends on its design.

Re-engineering of a database consists of a reverse-engineering process and a design process. During the design process, the derived conceptual schema is evolved, thereby, problems occur that also exist in database design.

It is desirable to support the database design process with tools to check design decisions or suggest improvements. Because of the abstraction process necessary to design a database, it is difficult to derive meaningful suggestions automatically. Reuse of existing databases can improve each of the tasks of database design. This article presents a method supporting the reuse of databases. It is organized as follows:

The overview presented in the next section outlining the main tasks of the reuse approach. In section 3, related works are enumerated. Section 4 presents a method for finding similar databases. In section 5, we derive design decisions from similar databases and adapt these onto an actual database. Section 6 discusses the necessity of a revise process. We then present two methods for organizing libraries for supporting the reuse process, and end with a summary.

2 Overview of the method

It is widely accepted that the case-based reasoning approach requires the following four tasks (originally suggested in [AaP94]):

RETRIEVE the most similar case
REUSE the information and knowledge in that case to solve the problem
REVISE the proposed solution
RETAIN the information likely to be useful for future problem solving

The key idea in reusing databases is similar to the idea behind case-based reasoning. We can adapt its tasks to the reuse of database design decisions, thereby resulting in the following tasks:

RETRIEVE the most similar database or part of a database
REUSE design decision for the actual database
REVISE the proposed design decision
RETAIN the information (e.g. build libraries suited to support the reuse)

Our approach assumes the following scenario. A set of existing databases is available, in addition to an actual database with incomplete structural, semantic, and behavioral information. We want to complete the database design, therefore, we search for a similar database among the set of existing databases in order to adapt design decisions.

Before we present our method, we provide an overview of some related work.

3 Related work

Storey/Chiang/Dey/Goldstein/Sundaresan: Database Design with Common Sense Reasoning and Learning. [SCD97] suggested a system supporting database design by using available databases. The approach emphasized determining similar pairs of attributes, entities, relationships, and applications. For this comparison, name information and an ontology aided in determining more complicated similarities, such as synonyms were exploited. Furthermore, a learning step of commonly valid databases for different applications is realized. These databases were then used to support the design of new databases. This method was tested with sample databases from well-known database literature.

Castano/DeAntonellis et al.: Schema Indexing, Clustering, Determining of Similar Databases. Several techniques relevant for reusing databases are described in numerous publications. [CAZ92], [CaA94], and [CaA96] suggested methods for determining the most important parts of a database (i.e. schema descriptors) by exploiting the number of paths, the number of attributes, and the hierarchy level of an object. The similarity between schemas was calculated by comparing schema descriptors ([CAZ92]) and by comparing all objects of the databases ([CaA96]). Schema abstraction based on schema similarity was described in [CaA94] and [CaA96].

Song/Johannesson/Bubenko: Finding Similarities for Schema Integration. [SJB96] deals with the problem of finding semantic similarities as a prerequisite for integrating schemas. The authors compared the meaning of entities and relationships by using "integration knowledge" containing information about synonyms and subset relationships. Attributes, key attributes, and cardinality constraints were also used to compare meanings of entities and relationships. This resulted in equivalent, compatible, and mergable schemas for use in the integration process.

Bergmann/Eisenecker: Reuse of Object-oriented Software. In [BeE95] the reuse of object-oriented software is realized as a type of case-based reasoning. The authors discovered that a method for determining similarities based solely on structural characteristics (names of methods, number and classes of parameters, and return value) returned poorer results than a method based on structural and semantic information.

4 Retrieve

If we want to reuse database design decisions, we first have to identify similar databases. This is a demanding task because databases are often complex and difficult to understand. We can only exploit available database characteristics (e.g. names, types, integrity constraints, transactions) for this task.

The process of comparing parts of databases is complex and is best realized using a bottom-up approach, thereby basing comparisons of complex concepts on comparisons of simpler concepts. Our approach begins with methods for finding similar attributes in two databases.

4.1 Determining similar attributes

This section presents heuristics for finding similar attributes. For each heuristic, a *similarity function* is evaluated for results between 0 and 1 (0 - no similarities, 1 - equality). The following database characteristics can be compared for delivering similar attributes:

H_a1 *Attribute names* (same names, same substrings in names, or synonyms)

H_a2 *Attribute types and lengths* (same or similar)

H_a3 Further structural information (e.g. *enumeration types, default values*)

These types of structural information suggest similar attributes. Nevertheless, their use will not determine all similarities because several homonyms and synonyms exist between two databases. [SJB96], [SCD97], and [CaA96] assume that synonyms are exploitable. Synonyms are domain-dependent, making it impossible to use a synonym dictionary for delivering correct results in every case. Although we believe that structural information aids in comparing databases, it is beneficial to include additional types of available characteristics.

When integrity constraints are already specified in the actual database and data are available, we can enumerate further heuristics:

H_a4. *Keys* (We determine if two attributes, A and B, are *keys* of their entities or relationships).

H_a5. *Functional dependencies* (If two attributes, A and B, appear to be similar, and the same *functional dependencies* are defined on these attributes, then this is an additional hint for similarity.)

H_a6. *Data* (same data values of two attributes)

If further characteristics of a database (e.g. behavioral information, transactions) are available, additional heuristics for comparing this information can be developed.

We now have some heuristic rules for indicating similar attributes. We subsequently compare and weight these heuristics. The more heuristic rules are fulfilled, the greater similarity measure should be. The following simple estimation can be used for this task:

$$sim(A, B) := \sum_{i=1}^{6} w_i * H_a i(A, B)$$

$H_a i$ - result of heuristic rule $\in [0..1]$
$w_i \in [0..1]$, $w_1 + .. + w_6 = 1$

The weights w_i can be determined using the following table specifying the reliability of the results of the enumerated heuristics. The more reliable heuristics shall be weighted higher than the less reliable rules.

very reliable	H_a1, H_a6
reliable	H_a2, H_a3
relevant only in combination with other rules	H_a4, H_a5

The *similarity of an attribute set* can be estimated in the following way:

$$sim(A_1..A_n, B_1..B_m) := max\left|\frac{2 * \sum_{i=1}^{n} sim(A_i, B_j)}{n + m}\right|$$

$j \in 1..m$, every j occurs only once

Based on these similar attribute sets, we begin the search for similar entities.

4.2 Determining similar entities

When searching for similar entities (E_1, E_2) in two databases (D_1, D_2), we employ a method based on rules for determining similar attributes ($A_{11}..A_{1n}$, $A_{21}..A_{2m}$) of the entities. Moreover, there are additional entity characteristic that can be included:

H_e1 *Entity names* (same names, substrings in entity names, or *synonyms*)

H_e2 *Keys* (same or different key attributes of two entities, E_1 and E_2)

Both heuristics deliver very reliable results, and can be assigned the same weight. These heuristics can be used in estimating similarity measures for entities:

$$sim(E_1, E_2) := \frac{1}{2}\sum_{i=1}^{2} w_i * H_e i(E_1, E_2) + \frac{1}{2} sim(A_{11}..A_{1n}, A_{21}..A_{2m})$$

$H_e i$ - result of heuristic rule $\in [0..1]$

$w_i \in [0, 1]$, $w_1 + w_2 = 1$

In this manner, the estimation of similar attribute sets enters into the calculation.

4.3 Determining similar relationships

When searching for similar relationships (R_1, R_2) in two databases (D_1, D_2), we can use the rules for determining similar entities (E_{11}, E_{12} and E_{21}, E_{22}) and similar attributes of the relationships($A_{11}..A_{1n}, A_{21}..A_{2m}$). Moreover, there are additional characteristics of the relationships that can be included:

H_r1 *Relationship names* (same names, substrings, and synonyms)

H_r2 *Keys* (same or different keys of two relationships, R_1 and R_2)

H_r3 *Same inclusion and exclusion dependencies*

H_r4 *Cardinalities* (when determining a similarity measure we include the similarity of entities. Additionally, we compare the associated cardinalities.)

The following table specifies the reliability of the results when using these heuristic rules. The weights w_i of the rules are derived from this overview:

very reliable	H_r1, H_r2
relevant only in combination with other rules	H_r3, H_r4

The similarity of two relationships can be estimated in the following way:

$$\begin{aligned} sim(R_1, R_2) := & \tfrac{1}{4}\textstyle\sum_{i=1}^{3} w_i * H_r i(R_1, R_2) + \tfrac{1}{4} sim(B_{11}..B_{1n}, B_{21}..B_{2m}) + \\ & \tfrac{1}{8} sim(E_{11}, E_{12}) + \tfrac{1}{8} H_r 4(R_1, E_{11}, R_2, E_{21}) + \\ & \tfrac{1}{8} sim(E_{21}, E_{22}) + \tfrac{1}{8} H_r 4(R_1, E_{12}, R_2, E_{22}) \end{aligned}$$

$H_r i$ - result of heuristic rule $\in [0, 1]$

$w_i \in [0, 1]$ 0..1, $w_1 + w_2 + w_3 = 1$

In this estimation, there are two possibilities for comparing the associated entities: E_{11} - E_{21} and E_{12} - E_{22} or E_{11} - E_{22} and E_{12} - E_{12} the one with the highest similarity measure is chosen.

4.4 Comparing more complex parts of databases

Different structural descriptions of two databases can have the same semantics. The same information is designed as an entity in one database can be defined as a relationship in another database. To find such cases, we compare entities with relationships. Therefore information about names, key, and belonging attributes can be used and weighted. Information about the similarity between an entity and a relationship is further used in the approach.

Same concepts could be represented in two databases with different granularity. For example, it may be that information represented by one entity in D_1, is represented by two entities and one relationship in D_2. We could find these similarities by evaluating the attributes and the names. The advantages of including such complex comparisons is that many more types of similarities can be found. However, it is more difficult to reuse information from such similar database parts because the adaption of design decisions is more complex. Furthermore, the search space increases. This method is mentioned here because it may be interesting for some applications, but, it is not used in our approach.

4.5 Building a graph

A typical problem occurring in the reuse of information is that one entity or relationship of a database may have similarities with several terms of another database. Consequently, we must determine which similar concept to choose for deriving further design suggestions. We apply a method known in graph theory as *graph matching* and use a bipartite weighted graph for representing the estimated similarities.

Definition ([Wes96]): *A* **bipartite weighted graph** *G consists of a non-empty vertex set* $V(G)$ *that can be divided into two disjoint subsets,* $S \dot{\cup} T$, *and a set of edges* $E(G) \subseteq \{(s,t) | s \in S, t \in T\}$. *All edges in* $E(G)$ *connect one vertex from* S *with one vertex from* T. *Every edge in the graph has a related weight.*

We can build a bipartite graph representing the estimated similarities as follows:

1. We begin with an empty graph.
2. For all determined similar entities and relationships ($V_1 \in D_1$, $V_2 \in D_2$), vertices are introduced (V_1 in S and V_2 in T). We draw an edge between these vertices. The weight of the edge is the determined similarity measure.

Next, we try to find a matching (a one-to-one relation) of the similar nodes with a maximal sum of all weights. Within a database context, this means that we search for similar parts of the databases ($D'_1 \subseteq D_1$, $D'_2 \subseteq D_2$: $D'_1 \approx D'_2$).

We demonstrate the suggested approach using an ongoing example consisting of two different databases designed for a university application (figure 1).

Figure 2 shows the bipartite graph that originates by overlaying the suggested method onto the two sample databases. We then determine a matching of this graph by constructing a cover, c, so that $c \geq w(M)$. This means, the cover is greater or equal to the weight of the matching. We then search for the case where the cover is equal to the weight of the matching. If this cover is constructed, we have determined a maximal matching ([Wes96]). Figure 2 presents the maximal matching for the similarity graph.

For the sample databases, we determine the weight of the matching which is $w(M) = 2.29$. This weight is subsequently used to choose the database most similar to an actual one.

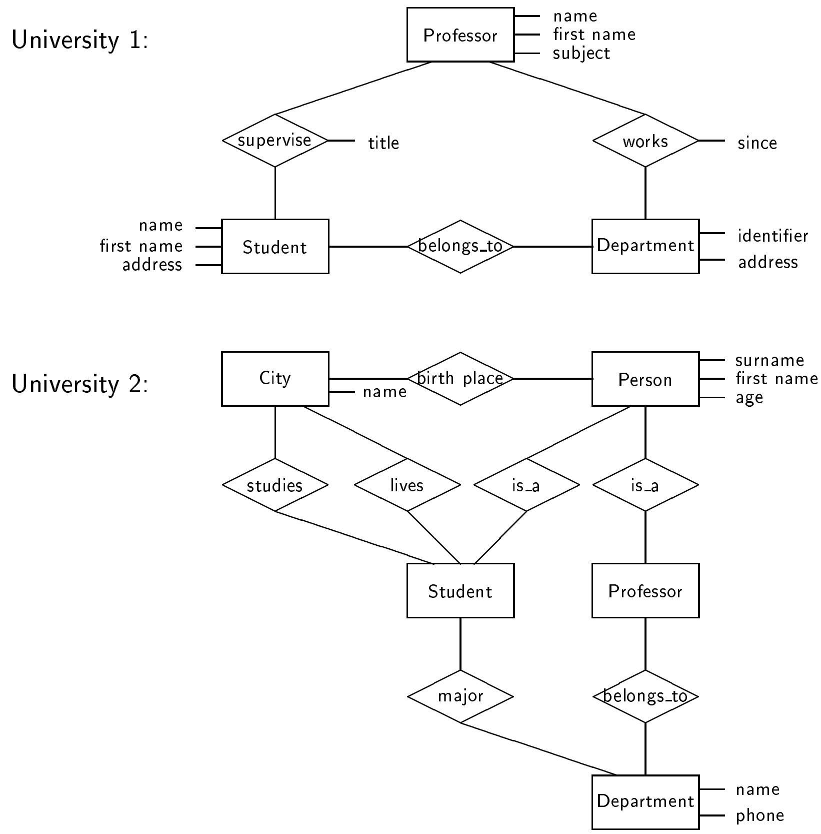

Figure 1: Sample databases

The resulting similar and dissimilar parts of the databases are shown in figure 3.

5 Reuse

We now have an actual database, and we have determined a similar database or similar part of a database. This section demonstrates which information can be reused for the actual database, and how this can be accomplished. Reuse of information from available databases can support the design, or redesign process, of a database.

5.1 Structural design

First, we demonstrate the kinds of structural completions and expansions that can be derived.

Addition of attributes. If there are similar entities or relationships, E_1 in D_1'

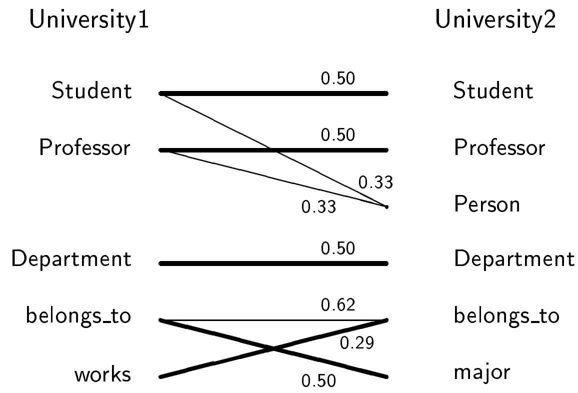

Figure 2: Maximal matching of the similarity graph

and E_2 in D'_2, and if E_1 contains attributes that are not in E_2, then we suggest adding these attributes into E_2.

Addition of path information. If there are two similar entities and relationships, E_1, R_1 in D'_1 and E_2, R_2 in D'_2, and if these are connected by a path, then we suggest adding this path information into D_2.

Addition of relationships. If there exists a relationship R_1 in $D_1 - D'_1$, and all associated entities of R_1 are in D'_1 (i.e. similar entities exist in the actual database, and the relationship doesn't exist in the actual database), then the relationship R_1 can be added in D_2.

Figure 3 illustrates such a case. The entities Student and Professor of University1 have similar entities in University2, but the relationship supervise does not. Therefore, we suggest adding the relationship supervise as an extension of the University2 database.

Addition of complex database parts. We suggest adding complex parts of the database D_1 into D_2, if similar entities E_1 in D'_1 and E_2 in D'_2 exist, and if E_1 has a direct link to nodes in $D_1 - D'_1$.

For example parts exist in the University2 database that are not in the University1 database (figure 3). Therefore, we suggest adding the concepts City, studies, and lives to extend the entity Student. In this way, we derive meaningful suggestions for an *inside-out design*.

5.2 Integrity constraints

Integrity constraints can also be derived from available databases by looking at the following points.

Functional Dependencies. If we determine similar attribute sets $A_{11}..A_{1n}$ in D'_1, $A_{21}..A_{2n}$ in D'_2, and if a functional dependency $A_{1i} \longrightarrow A_{1j}, i, j \subseteq 1..n$ is valid, then the corresponding functional dependency in D'_2 could also be valid.

Keys. If similar entities E_1 in D'_1, E_2 in D'_2, and similar attributes $A_{11}..A_{1n}$,

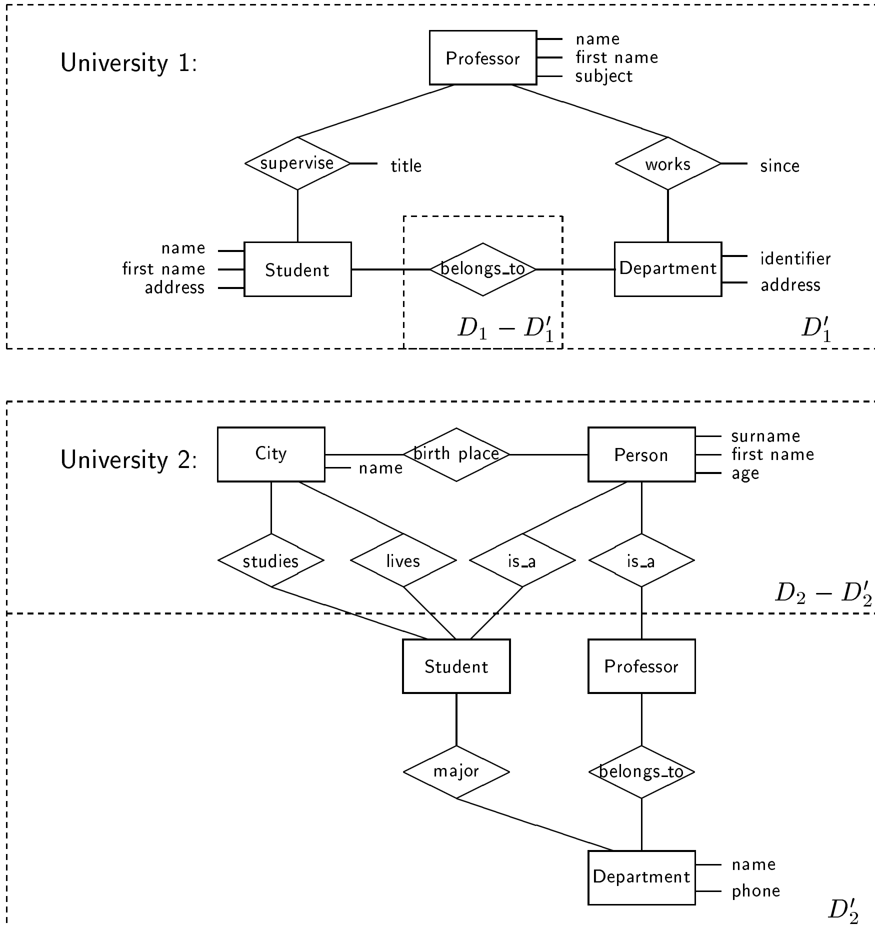

Figure 3: Similar and dissimilar portions of the sample databases

and $A_{21}..A_{2n}$ are derived, and if the attributes $A_{11}..A_{1n}$ are a key of E_1, then $A_{21}..A_{2n}$ could also be a key of E_2.

Candidate keys for relationships are determined in the same manner.

Inclusion and Exclusion Dependencies. If we find two databases with similar entities or relationships and similar belonging attribute sets, and an inclusion or exclusion dependency is defined on the attributes of D_1, then the corresponding dependency may also be valid in D_2.

Cardinality Constraints. If two databases with similar entities and relationships E_1, R_1 in D_1' and E_2, R_2 in D_2' exist, and the cardinality constraint $\mathsf{card}(R_1, E_1)$ is fulfilled in D_1', we can also expect $\mathsf{card}(R_2, E_2)$ in D_2' to have the same value.

5.3 Behavioral Information, Sample Data, Optimization, and Sample Transactions

There are several additional characteristics, that can be used in the same manner.

If behavioural information is formally specified in available databases, then we can reuse this specification in similar databases. Furthermore, sample data, suggestions for optimization, and sample transactions can be reused.

6 Revise

Reusable characteristics only provide suggestions and must be revised in some way.

Some design decisions can be checked *without the user*. For example, suggested integrity constraints can be checked to determine they do not conflict, and that they conform with sample data.

Other suggestions have to be discussed *with the user*. If the reuse approach is used to derive suggestions for design tools, and a user is required to confirm the decisions, this demand is fulfilled.

For integrity constraints, a method was demonstrated in [Kle97] that acquires candidates for integrity constraints by discussing sample databases.

7 Retain

We have shown that the tasks retrieve, reuse, and revise can suggest design decisions for a database. To apply the method, we compared complete databases. We now demonstrate two methods for organizing the databases into libraries to support the reuse process more efficiently.

7.1 Determining necessary and optional parts

In [SCD97], the similar parts of databases are stored for further use. In our approach, we store the parts occuring in every database *for a field of application* and the distinguishing features of the databases. Several points must be discussed with the user, who must decide which one of the synonym names of two similar databases is the most suitable. Further, the user has to confirm which attributes of an entity or relationship are relevant for a common case.

In any event, the derived parts must be extended so that all databases are complete. This means if a relationship exists in the database, and not all associated entities are in this database, then the entities have to be added. The positive side-effect of this action is that the database parts could be integrated based on these entities that now occur in the similar and dissimilar portions.

If users want to design a database for one field of application, then they can be supported as follows:

1. The part occuring in every database is discussed. If it is relevant to an actual database, then we continue.
2. The distinguishing features of the existing databases are discussed. If they are relevant, then they are integrated into the actual database.

7.2 Deriving database modules for the reuse process

A second method for developing libraries is dividing a database into modules. The question then arises of how to locate these modules. For this purpose, the following available information is exploited:

- path information
- integrity constraints (e.g. inclusion dependencies, foreign keys)
- the course of design process (e.g. which concepts are added together)
- similar names
- available transactions
- layout (if not automatically determined)

All these characteristics determine how closely entities and relationships belong together. A combination of these heuristic rules can be used to determine clusters in a database. These clusters are stored as units of the database, and the reuse process is based on these units.

8 Conclusion

The method presented in this article relies on heuristics and an intuitive way of weighting these heuristics. Therefore, we cannot guarantee that it always delivers correct results, however, the method is simple, easy to apply, and promising for many design decisions. One of its advantages is that many different database design tools can use the same method.

This method was developed as a part of a tool for acquiring integrity constraints [Kle97]. The tool's main appoach was to realize a discussion of sample databases and to derive integrity constraints from the users answers. Thereby, the approach presented in this article was one method to derive probable candidates for keys, functional dependencies, analogue attributes, inclusion and exclusion dependencies. The semantic acquisition tool was developed within the context of the project RADD[1].

Another practical application of the method will be embedded in the GETESS[2] project that focus on the development of an internet search engine. Thereby, we design databases for storing the gathered information by using ontologies. Because ontologies are domain-dependent and for every domain an own ontology is necessary, existing ontologies and the corresponding databases shell be reused. The ontologies resemble conceptual databases, therefore, we can adapt the demonstrated method for use in this project.

---

[1] RADD - *R*apid *A*pplication and *D*atabase *D*evelopment Workbench [AAB95], was supported by DFG Th465/2

[2] GETESS - *GE*rman *T*ext *E*xploitation and *S*earch *S*ystem [SBB99] supported by BMBF 01/IN/B02

9 Acknowledgement

I would like to thank Bernhard Thalheim very much for encouraging me to publish this paper.

References

[AAB95] Meike Albrecht, Margita Altus, Edith Buchholz, Antje Düsterhöft, Bernhard Thalheim: *The Rapid Application and Database Development (RADD) Workbench — A Comfortable Database Design Tool.* In 7th International Conference CAiSE '95, Juhani Iivari, Kalle Lyytinen, Matti Rossi (eds.), Lecture Notes in Computer Science 932, Springer Verlag Berlin, 1995, pp. 327-340

[AaP94] Agnar Aamodt, Enric Plaza: *Case-based reasoning: Foundational issues, methodological variations, and system approaches.* AI Communications, 7(1), 1994, pp.39-59

[BeE95] Ralph Bergmann, Ulrich Eisenecker: *Fallbasiertes Schließen zur Unterstützung der Wiederverwendung objektorientierter Software: Eine Fallstudie.* Proceedings der 3. Deutschen Expertentagung, XPS-95, Infix-Verlag, pp. 152-169, in German

[CaA94] Silvana Castano, Valeria De Antonellis: *Standard-Driven Re-Engineering of Entity-Relationship Schemas.* 13th International Conference on the Entity-Relationship Approach, P. Loucopoulos (ed.), Manchester, United Kingdom, LNCS 881, Springer Verlag, December 1994

[CaA96] Silvana Castano, Valeria De Antonellis, Carlo Batini: *Reuse at the Conceptual Level: Models, Methods, and Tools.* 15th International Conference on Conceptual Modeling, ER '96, Proceedings of the Tutorials, pp. 104-157

[CAZ92] Silvana Castano, Valeria De Antonellis, B. Zonta: *Classifying and Reusing Conceptual Schemas.* 11th International Conference on the Entity-Relationship Approach, LNCS 645, Karlsruhe, 1992

[Kle97] Meike Klettke: *Akquisition von Integritätsbedingungen in Datenbanken.* Dissertation. DISDBIS 51, infix-Verlag, 1997, in German

[SBB99] Steffen Staab, Christian Braun, Ilvio Bruder, Antje Düsterhöft, Andreas Heuer, Meike Klettke, Günter Neumann, Bernd Prager, Jan Pretzel, Hans-Peter Schnurr, Rudi Studer, Hans Uszkoreit, Burkhard Wrenger: *GETESS — Searching the Web Exploiting German Texts.* Cooperative Information Agents III, Proceedings 3rd International Workshop CIA-99, LNCS 1652

[SCD97] Veda C. Storey, Roger H. L. Chiang, Debabrata Dey, Robert C. Goldstein, Shankar Sundaresan: *Database Design with Common Sense Business Reasoning and Learning.* TODS 22(4): pp. 471-512 (1997)

[SJB96] William W. Song, Paul Johannesson, Janis A. Bubenko: *Semantic similarity relations and computation in schema integration.* Data & Knowledge Engineering 19 (1996), pp. 65-97

[Tha99] Bernhard Thalheim: *Fundamentals of Entity-Relationship Modelling.* Springer Verlag, 1999

[Wes96] Douglas B. West: *Introduction to Graph Theorie.* Prentice Hall, 1996