AN INFRASTRUCTURE APPROACH TO DARTABASE COMPONENTS

ROLAND KASCHEK
Department of Information Systems Massey University
Palmerston North, New Zealand
R.H.Kaschek@massey.ac.nz

ABSTRACT

In large scale database development the data in a database to be developed is not raw data. Rather it is extracted from other databases. Large scale database development therefore would benefit from employing database components, as these would have the potential for simplifying database reuse. In this paper the concept of database components is discussed. The approach used to database components is based on a particular infrastructure. That infrastructure aids in providing and using database components and may be used for establishing a component market. The idea to use such an infrastructure results from under5standing database components and database reuse as a problem of software economy rather than a technical or motivational problem. The infrastructure helps establishing a component- and database- reuse culture. It enables to store so-called component candidates. These may be promoted to database components if usage experience indicates so. Similarly, a database component may be relegated to a component candidate if that appears as reasonable.

Keywords component, database, CBD, modelling, software economy, data engineering, software engineering

1 INTRODUCTION

Information systems development requires creation and maintenance of a database that stores the data that may be queried and then used by users of the information system. Relational databases still count for the majority of databases used. So far their largest building block is the table. For small-scale database development where a database is developed that stores and maintains raw data this is sufficient. For large scale development where databases have to be developed on top of other databases and source in some of their data that might not be so. For this kind of database development the availability of larger database building blocks would simplify the development process and thus increase the quality of the information system utilizing the database. Large scale database development is not so much a problem of size as it is of dynamically integrating available databases that have their own refresh-cycle and database administration.

Database components shall aid in developing new databases out of sets of old databases, i.e., components. Some "glue" will be needed to fit everything together. This glue on the one hand will simply be based on conventional relational concepts, in particular "foreign keys" and "joins". On the other hand non-relational operations that will be defined on components serve as that glue. Component based development (CBD) shall improve system quality and at the same time reduce system development-cost and -time. The author believes that database components as defined here, if used, will simplify systems integration, development and maintenance, increase the reuse of systems or parts thereof, and will therefore increase data quality, a major information system quality aspect. Also buying and selling of databases can be simplified by using database components. Traditionally the trading of databases is limited to, e.g., phone- or address books, in particular yellow pages and the like. Also the various schedules such as flight plans, sailing lists, train- and bus-schedules are directly part of business deals. Finally, dictionaries are databases that are traded traditionally.

In this paper the structure of a database is not focused at. Rather the database's utility is used as an indicator for whether that database should be a component. The well-known two-library approach, is employed for devising an infrastructure as well as a respective usage model that allows dealing with database components. For the respective infrastructure database components essentially will be databases defined on top of databases. They will be introduced by decision of an approval committee and then stored in a repository. Component candidates that do not make their way into the component repository will be stored in candidate repository. Jury decisions govern the flow of component candidates into the component repository and the flow of components back into the candidate repository. While using components will be up to the disposal of a development team at hand candidates can be used for systems development only after approval by the jury. Using either of them might be due to a usage charge.

Ascribing a component candidate the status of component is a jury decision that not only considers formal properties of the candidate. The reason for this lies in the fundamental distinction in the class of mappings as either synthetically or empirical. An empirical mapping is a mapping for which a check of correctness of association of ordinate values (i.e., values of the depending variable) to abscissa values (i.e., values of the independent variable) cannot be done formally. A mapping is here called synthetically if such check can be done formally, i.e., using computational resources only. The correctness check of the mapping

that associates to employees their date of birth in the end involves reference to birth certificate and identity card. Therefore it is empirical. On the other hand checking whether a particular mapping correctly associates the square of a number to that number involves a mathematical proof. It is the view of this paper that software engineering in part is distinguished from data engineering by software engineering ultimately aiming at identifying and implementing (by means of algorithms) suitable synthetically mappings while data engineering aims at identifying and implementing (by means of databases) empirical mappings. Note that the distinction between synthetically and empirical mappings is not one of representation, as all mappings can be represented as one-tuple-table with the table heading containing the abscissa values and the tuple providing the ordinate values. The distinction reflects the existence of a mathematical proof of correctness of the ordinate-to-abscissa association provided by a given implementation of a mapping.

If a number of important problems with respect to a domain can be solved with the help of the empirical mappings in a database it can be expected that this database will be considered worth the necessary investment. In that case an inter-organizational or public market for that database can be established. Assessing the utility of a database with respect to a given domain is a decision that relies on expert knowledge of the domain in question.

2 RELATED WORK

Large scale database design appears to be connected with data warehousing. Searching with the search engine “Google” for definitions of that term reveals that a number of quite different definitions is used in the Web. (A search for the key words "define: ’Data Warehouse’" done on 15 February at 12:45 am resulted in 25 definitions of the term provided.) Several of the definitions found shared some of the specifications provided by the one quoting William Immon as saying that a data warehouse is a "Subject-Oriented, Integrated, Time-Variant, Nonvolatile collection of data in support of decision making". The definition continues with "Data Warehouses tend to have these distinguishing features: (1) Use a subject oriented dimensional data model; (2) Contain publishable data from potentially multiple sources and; (3) Contain integrated reporting tools." (see www.peaksoftware.com/glossary/) Most of what is addressed in this definition does not apply to database design in the large, as understood in this paper. Data warehousing therefore is not considered any further in this paper.

Based on Wieringa, [Wie03], within this paper a component is a part of an information system that delivers a service to its environment. A component is here considered as empirical or synthetically if the service it delivers is empirical or synthetically respectively. Following Herzum and Sims [HS00] it is required here that a component is sufficiently self-contained, i.e., can be deployed and plugged into an information system by considering defined interfaces only.

The services delivered by a component can be classified as primary or secondary. Primary services are those for implementation of which the respective information system was built. Secondary services are those that can aid in identifying, finding, and using primary services. With respect to a database both classes of service are important and well-known. The primary service is provided by a database programming language and allows recording, storing, manipulating, and retrieving data, i.e., empirical mappings. The secondary service is provided by the self-description facility, i.e., the catalogue of the database that enables it to provide users on overview of the database structure, i.e., its schema.

Thalheim [Tha03] has, based on the work of Broy, [Bro97], started discussing the component concept for databases. He focuses on secondary services, i.e., schema components, as he aims at a schema engineering based on schema components. This paper focuses at primary services, i.e., database components. It seems to be that not much work has been done in this respect. There is, however, Notess’ "Google Special Report: Database Components" in the Search Engine Showdown (accessed from http://www.searchengineshowdown.com/features/google/dbanalysis.shtml at 10 Feb 2005). This report does not aim at a structure analysis. Rather it list the ratios of the various kinds of contents the search engine Google had in its database by 4 - 6 March 2002. A theoretical investigation into understanding database components in the sense of this paper is not intended by Notess. The author believes that schema components cannot be successful in practice if not accompanied by a concept of database component. A well-known database text book such as Elmasri & Navathe [EN94] does not have an index entry for "schema component", for "database component", and for "component".

Database design in the large (that here is understood to be a conceptual design) is different from conventional conceptual database design in so far as it explicitly takes into account already existing databases, which is not the case for conventional conceptual database design. Batini et al., for example, say: "The objective of conceptual database design is to produce a high-level DBMS-independent conceptual schema, starting from requirement specifications that describe reality...", see [BCN92].

3 A LARGE SCALE DEVELOPMENT EXAMPLE

In a large company, such as a full-scale internationally operating bank, the individual departments may have constructed large databases the schema of which is remarkable stable over time. There will be, however, versions of it and in rather regular time-intervals new current extents. These databases may be of interest for a lot of the data processing in the company and thus actually are heavily reused. The example discussed here involves the hypothetical biggest bank of Switzerland (BBS). It is presupposed that for provisioning of data services regarding risk assessment and (based on it) enterprize steering a new information system (CFS, i.e., component fabrication system) shall be introduced. CFS shall dynamically integrate the available data and bring it into the shape needed by business analysts and top management. The clients of CFS often will not pay much attention for quality aspects such as lack of redundancy that are important for storing and maintaining data. Rather they want data in a form that can be analyzed easily. This requirement in some cases even might enforce de-normalization and thus introduce redundancy. The same goes for databases sourced from CFS, as obtaining the data is the primary concern of the respective vendors. Maximum independence of the databases provided by CFS from the sources on which it builds had to be achieved. Therefore it was decided that CFS copies the databases it needs as input and locally maintains these copies. Similarly customers of CFS may ask for copies of the databases they need. Obviously the files sent to CFS as well as those sent from CFS, as they contain the databases, may be quite large. Storing the input databases as local copies enforces CFS to update these in a controlled way. Consequently CFS' output databases must be updated accordingly. Note that based on the peculiarities of the business in which CFS's data providers and data consumers are involved in respectively the refresh period applied to the individual databases may be different.

Assume now that CFS has a request to deliver to the RIS (risk information system) on a regular base the last month's provisions, writeoffs and recoveries. A preliminary business analysis has shown that CSF will need data according to the high level ER-diagram in Figure 1 (see the appendix below). In this figure dashed lines signify the department of BBS, from which the data described by the entity types inside the dashed lines can be retrieved (KDB, CAS and IS respectively stand for "Kundendatenbank" (i.e., customer database), "Critical Assets System", and "International Services"). A more sophisticated business analysis has shown that the data records described by "Kunde" (i.e., customer) can be retrieved from IS and actually are subsumed under the data records described by "Partner" from IS. Similarly, it resulted from a further business analysis that the data records described by "Konto" (i.e., account) including the association to "Kunde" can be retrieved from IS as well. Consequently, rather than retrieving data from all three departments mentioned in the Figure 1 it suffices to use two of them, i.e., CAS and IS. Clearly the decision to use only two different sources as input results in fewer dependencies to be observed, higher data quality and higher availability of the data CFS was requested to deliver.

In the example the infrastructure that is proposed below is not in place. However, one can imagine that based on the departmental maintenance of their local databases such infrastructure could be introduced. Retrieving departmental data then would be limited to accessing components in the component repository. This first of all would centralize all available components and any staff looking for available data would know where to have a look at. Furthermore, the functions of collecting and maintaining data as it is naturally done in departments would be separated from economically managing the data and therefore could be implemented more efficiently. Finally, introducing a work fore role that cares about managing database components would help establishing and systematizing the knowledge for effectively and efficiently use database components.

In the example no detail is provided regarding the attributes of the entity types shown in Figure 1. It is, however, clear that the decision as to whether the data records described by "Kunde" can be subsumed under the data records described by "Partner" is not a formal one. In particular it cannot be decided by only referring to the database schema. The definitions of the respective attributes need to be read, analyzed and compared. Furthermore the established practice of including a particular tuple in one of the mentioned tables needs to be discussed. That discussion needs to disclose the availability of the respective empirical mappings. The discussion below is restricted to the case of relational databases. The author believes that this does not cause difficulties to readers.

4 RELATIONAL DATABASES

For sets M , N a mapping f: $M \rightarrow N$ is a right-unique relation $f \subseteq M \times N$ and for such mapping the term def(f) denotes the set $\{ m \in M \mid \exists n \in N, (m, n) \in f \}$. Let B be a set of so-called base types, i.e., of pairs t = (n , e), where n is the name of the base type and e is its extent, i.e., the set of values of this type. Frequently occurring base types are, e.g. INT, CARD, and STRING respectively having implementable subsets of the sets of integers, non-negative integers and strings over a given alphabet as value sets. A **relation schema** S over B is a 5-tuple $(N, \Gamma , \Pi , \Phi , \Delta)$ such that:

- N is the **name** of the relation schema,

- $\Gamma = \{ (c_1 , b_1) , \dots , (c_m , b_m) \}$, such that *m* is a non-negative integer and a subset $\{ (n_1 , b_1) , \dots , (n_m , b_m) \} \subseteq B$, exists. Γ is called the **type** of S. Each element (c , b) of Γ is called **attribute** of *S* and *c* is called the **name** thereof.
- $\Pi \subseteq \Gamma$, which is called the **primary key** of *S*,
- Δ is a set of logical formulae, the so-called **constraints** of *S*, and
- Φ is a set of triples (R , T , f), such that $R \subseteq \Gamma$, T is a relation schema with primary key Π_T, f $\in$ STRING, and a bijection $\beta : R \rightarrow \Pi_T$ exists such $\forall$ (r , b) $\in$ R $\exists$ (p , b) $\in \Pi_T$, with β(r , b) = (p ,b). Each element (R , T , f) of Φ is called **foreign key** of *S* on *T* if $S \neq T$ and **self reference** otherwise. The string f is called the **role** of T in S.

When there is no doubt regarding the set of base types over which a relation schema is considered or, when it is not important for the purpose at hand which set of base types is actually presupposed, then one simply uses the term *relation schema* rather than relation schema over B

Let S be a relation schema over base types B. A **relation** R over S is a pair (N , Ω), where N $\in$ STRING is the name of the relation and Ω is a set of partial mappings $\varpi : \Gamma \rightarrow \cup_{(c , b) \in \Gamma}$ b, called **extent** of the relation schema S, such that the following assertions hold:

- $\Pi \subseteq$ def(ϖ), $\forall \varpi \in \Omega$,
- ϖ(c , b) $\in$ b, $\forall \varpi \in \Omega$, (c , b) $\in$ def(ϖ),
- $\varpi|_\Pi = \varpi'|_\Pi$ implies $\varpi = \varpi'$ and Π is minimal with this property.
- Ω is a model of Δ, i.e., all formulae in Δ are true when interpreted in Ω.

A **database schema** $\sum$ is a finite set $\{S_1 , \dots , S_o \}$ of relation schemas $S_i = (N_i , \Gamma_i , \Pi_i , \Phi_i , \Delta_i)$, i $\in$ {1,...,*o*}, such that T $\in \sum$ holds, $\forall$ (R , T , f) $\in \Phi_i$, i$\in$ {1,...,*o*}. A **database** D over the database schema $\sum$ is a triplet (N_D , R , d), such that N_D is the name of the schema, $R = \{R_1 , \dots , R_o \}$ is a set of relations $R_i = (N_i , \Omega_i)$ over S_i , for all i $\in$ {1,..., *o*}, and d is the as-of date, i.e., the date at which the empirical mappings in the extents of the schemas involved in the database are supposed to be correctly describing the state of affairs. It furthermore is required that the following **foreign key assertion** holds: $\forall$ i $\in$ {1 ,..., *o* }, (R , S_j , f) $\in \Phi_i$, $\varpi \in \Omega_i$ $\exists \varpi' \in \Omega_j$, such that ϖ(c , b) = $\varpi'(\beta_{S_i S_j}$(c , b)) , for all (c , b) $\in$ R. Here $\beta_{S_i S_j}$ is the bijection introduced in the definition of the term relation schema. Note finally that the full complexity of treating temporal aspects of data are ignored here. In more sophisticated banking applications, however, these have to be considered to a larger extent. A legal or managerial requirement is to be capable of reproducing important documents that heavily depend on the state of basic data at any future point in time. The so-called bi-temporal data storage was introduced, see for that requirement, e.g. [Kün04]. In that way of storing data additionally to the correctness-dimension of time that was considered above the awareness-dimension of time is introduced.

The Figure 2 (see appendix below) shows a database with the two relation schemas "Employee" and "Project". Assume that these respectively are the names of the mentioned relation schemas. The database contains relations R_1, R_2 over "Employee" and "Project" respectively. The Employee" schema has the attribute set Γ={ (n_1,STRING), (n_2, STRING), (DOB, DATE), (p, STRING), (q, STRING) }. Its primary key P_i is the set of the three attributes underlined twice, i.e., {(n_1,STRING), (n_2,STRING), (DOB, DATE) }. Its set of formulae Δ is empty and its foreign keys are indicated by underlining, i.e., ((p, STRING), Project, leads), and ((q, STRING), Project, works). The roles of the relation schema “Project” in these foreign keys are “leads” and “works”. The relation schema “Project” can be analyzed accordingly. The main difference is that it does not have any foreign keys. The relation R_1 comprises the set of the following partial mappings

{< (n1, STRING), John > , < (n2 , STRING) , Smith > , < (DOB , DATE) , 1/1/1984 > , < (q , STRING) , red > },

{ < (n_1 , STRING) , Jane > , < (n_2 , STRING) , Jones > , < (DOB , DATE) , 20/5/1985 > , < (p , STRING) , red > },

{ < (n1 , STRING) , Jim > , (n2 , STRING) , Brown > , < (DOB , DATE) ,12/3/1978 > , < (p , STRING) , blue > }.

The elements of the relation R_2 can as well be determined easily.

4.1 DATABASE COMPONENTS

The term database component is not formally defined in this paper. It rather only formally defines the term database component candidate and leaves it up to a jury to promote a candidate to a component or to relegate a component to a candidate if that appears to be suitable. The concept of component in this paper thus is a fully pragmatic one. Component candidates meet the formal requirements for being components. Whether such candidate actually becomes or stays a component depends on the experiences that the organization makes with using the service provided.

A **database component candidate** C or simply **candidate** is a 4-tuple (N , E , I , q), such that N is the **name** of C, E is the **exported database**, I is the **imported database**, and q is a surjective partial

mapping q : $\cup_{R\in I}$ R $\rightarrow$ $\cup_{R\in E}$ R, the so-called **defining query**. In this definition the view concept (i.e., defining query) was used for defining the concept of candidate. Note that the limitation to just one input database does not restrict expressivity since the union of a set of databases can be considered as a database. Taking the union over a set of databases only reduces the refreshment cycle of the union to the minimum refreshment cycle of the databases in that set.

Obviously databases can be considered as special case component candidates where the imported database equals the exported database and the defining query is the identity. The definition of component candidate shows that database components in fact can be considered as black boxes implementing particular empirical mappings as is required by the component literature. Additionally to stating what a component candidate (and thus a component) is we introduce two different ways of using components for defining databases. These ways are component composition and database specialization. Database specialization applies to components in so far as it can be applied to the exported as well as to the imported database of a component.

Let candidates C_1 , C_2, C_3 be given for i $\in$ {1, 2 , 3 }. Let further be $E_3 = I_2$, $I_3 = I_1$, $I_2 = E_1$ and $q_3 = q_2 \circ q_1$, then C_3 is called the **composition** $C_2 \circ C_1$ of C_2 and C_1. The composition of candidates is obviously associative.

Let i $\in$ {1 , 2 }, m_j an integer, $j_i \in$ { 1 , … , m_j } and $S^i_{j_i}$ = ($N_{i,j}$, $\Gamma_{i,j}$, $\Pi_{i,j}$, $\Phi_{i,j}$, $\Delta_{i,j}$) be a relation schema, $R_{i,j}$ = ($N_{R_{i,j}}$, $\Omega_{i,j}$) a relation over $S^i_{j_i}$ and $\sum_i$ = {S^i_1 ,…, $S^i_{m_i}$ }, a database schema, and finally D_i a database over $\sum_i$. Then $\sum_2$ is called specialization of $\sum_1$ if there is an injective mapping ι : $\sum_1 \rightarrow \sum_2$, such that ι (S) is a specialization of *S*, for all S $\in \sum_1$. Let ι (S) = (N' , Γ', Π' , Φ' , Δ'), and S = (N , Γ, Π , Φ , Δ). Then ι (S) is a specialization of S, if $\Gamma \subseteq \Gamma'$, $\Pi \subseteq \Pi'$, $\Phi \subseteq \Phi'$, and $\Delta \Rightarrow \Delta'$ hold. Finally, let $\sum_2$ be a specialization of $\sum_1$. Then D_2 is called specialization of D_1 if regarding the specialization mapping ι each relation $R_2 = (N_2 , \Omega_2)$ in D_2 over ι (S , Γ , Π , Φ , Δ) = (N' , Γ', Π' , Φ' , Δ') is a specialization of the relation $R_1 = (N_1 , \Omega_1)$ in D_1 over (S , Γ , Π , Φ , Δ). That is the case if $\Omega_2|_{\Gamma \setminus \Gamma'} = \Omega_1$ holds. Obviously specialization is a reflexive and transitive relation on the class of all databases.

Once a number of component candidates is available for use a respective infrastructure and usage model will be needed. Both of these are discussed below. The intended mechanisms of derivation of new components from old ones are composition or specialization. Both of these operations reuse already cleansed and validated data. Compared to following a green field approach an increased data quality and reduced project duration can thus be expected. Note that for large organizations such as BBS one must expect to deal with really large databases the schema of which may have 2000 relation schemas and 15000 attributes, [Kün04]. Also the transaction load must be expected to be very high, i.e., above in total 1,000,000 transactions per hour in average for the key databases (general ledger, master & reference data, account assignment logic, valuation logic, and the front system databases) ,[Kün04]. It is obvious that reusing the primary or secondary services of such databases requires database building blocks above the relation level. A glimpse of an idea of how computer use in Swiss banks came to happen can be obtained from Neukom, [Neu04].

5 A COMPONENT INFRASTRUCTURE

Database components shall simplify development and maintenance of information systems and in particular aid in increasing their data quality. These requirements appear to depend on the context of component creation and use. It therefore is unlikely that a useful component concept can be defined based on the structure of databases only. Rather it appears likely that a usage model and an infrastructure should be proposed that helps organizations to identify, create, use and maintain database components. Such a usage model and respective infrastructure idea is borrowed from the software reuse community, see, e.g., [Gra98].

The infrastructure consists of a component repository *R*, a candidate repository *C* and a connection *P* between these. The items stored in *R* or *C* are called components and candidates respectively. The infrastructure allows items being moved from *R* to *C* and vice versa. An organization may use this infrastructure such that:

1. A component jury is implemented that has authority about the candidates to be stored in *R* and *C* and that awards an initial score to these data. That jury furthermore defines discrimination threshold *R*-, *C*+ and *C*- for components and candidates respectively such that:
    - a component the score of which falls under *R*- is flagged as candidate for being moved from *R* to *C*,
    - a candidate the score of which grows over *C*+ is flagged as candidate for being moved from *C* to *R*,
    - a candidate the score of which falls under *C*- is flagged as candidate for removal from *C*
2. Every staff may propose candidates being added to *C*.
3. Every staff may propose candidates or components being awarded a score increment or decrement.

4. The jury decides on the initial population of *C* and the initial score of each of its inhabitants.
5. The jury defines the score increments or decrements awarded to inhabitants of *R* and *C*.
6. The jury decides about what to do with data collections in *R* or *C* the score of which has fallen under *R*-,*C*- or has grown over *C*+.
7. Components are free for use by everyone. Candidates may only be used after Jury approval.

For each component or candidate respectively in *R* or *C* there is some additional information stored with the data. For example, for each candidate and for each component a generic definition is stored that allows to decide whether an empirical function is an instance of the candidate or component or not. Furthermore for each relation *R* in the exported database that is a relation over the schema $S = (N, \Gamma , \Pi , \Phi , \Delta)$ a list of synonyms to the defined attributes in Γ as well as a list of related terms is stored and can be queried. Similarly for each attribute a generic definition is stored according to which a function value $\varpi (c , b)$ can be defined provided for the primary key Π the function values $\varpi (p , b)$ are known, $\forall (p , n) \in \Pi$. Also a synonym list and related terms are stored for each column.

6 IDENTIFYING POTENTIAL FOR COMPONENT-REUSE

The potential for reusing a component depends on finding a universe of discourse (in ones own environment) that is sufficiently similar to the database exported from the component so that one can hope to adapt the component at sensible cost to what one actually needs. Two relations between databases have been defined above that could be used in this respect, i.e., composition and specialization. Obviously a suitable modification of the defining query of a component comes into account even if none of these relationships applies.

For making the terminology more precise in the sequel the term universe of discourse of a database is defined. Whether a domain that in the sense of the following definition is a universe of discourse exists physically or only conceptually is of no relevance. Let $\sum = \{S_1 , \dots , S_o \}$ be a database schema and $D = (N , R , d)$ be a database over $\sum$ with $R = \{ R_1 , \dots , R_o \}$ a set of relations such that $R_i = (N_i , \Omega_i)$ is a relation over relation schema $S_i = (N_{S_i} , \Gamma_i , \Pi_i , \Phi_i , \Delta_i)$ with $\Gamma_i = \{ (c^i_1 , b^i_1) , \dots , (c^i_{m_i} , b^i_{m_i}) \}$, for all $i \in \{1,..., o\}$. A set *U* is called **universe of discourse** of D if there exists a bijection $a : U \rightarrow \cup_{i \in \{1,..., o\}} \Omega_i$, such that the following assertions hold.

1. For all $i \in \{1,..., o\}$ there is a thing-predicate t_i defined on U, such that $t_i(u) \equiv$ true holds iff $a(u) \in \Omega_i$.
2. For all $i \in \{1,..., o\}$ there is a property-predicate $p_{i,j}$ defined on U such that for $u \in U$ with $t_i(u) \equiv$ true it holds that $p_{i,j} (u , x) \equiv$ true iff $a (u) (c^i_j , b^i_j) = x$.
3. $u = v$, iff the following assertions hold true
   (a) $t_i(u) \equiv$ true iff $t_i(v) \equiv$ true , for all $i \in \{1,..., o\}$.
   (b) $a (u) = a (v), \forall i \in \{1,...,o\}$ with $t_i (u) \equiv$ true $\equiv t_i (v)$.
4. A reference predicate r is defined on U such that $r (u , f , v) \equiv$ true, if $t_i (u) \equiv$ true $\equiv t_j (v)$, $(R , S_j , f) \in \Phi_i$, and $a (u) (c , b) = a (v) (\beta_{S_i S_j} (c , b))$ holds, for all $(c , b) \in R$.

The predicates introduced right now define a rudimentary language for accrediting properties to entities. A business expert who is educated to use it can after studying the available components come up with a proposal for using such a language to design a solution for a problem at hand. That indicates how the proposed infrastructure could help in reusing databases. Respective experts would know what components are available and would match a problem at hand to the database that allows for the most suitable problem solution.

Obviously the set $\cup \Omega_i$ of all partial mappings in the database is a universe of discourse of database D. The purpose of modelling languages like the Entity Relationship Model, [Che76], in data engineering is the creation of a universe of discourse of a database in a way that easily a respective database can be defined.

Glass says in [Gla03] that "(r)euse in the large remains an unsolved problem, ..." The proposed infrastructure and usage model of repositories to some extent allows people to try out what can be reused and is helpful and what is not. One has, however, to observe that there is no "free lunch". Or, as Glass puts it in [Gla03] "(t)here are two 'rules of three' in reuse: (a) It is three times as difficult to build reusable components as single use components, and (b) a reusable component should be tried out in three different applications before it will be sufficiently general to accept into a reuse library." And finally: "Modification of reused code is particularly error-prone. If more than 20 to 25 percent is to be revised, it is more efficient and effective to rewrite it from scratch." [Gla03]. One therefore should be careful regarding the expectations in reuse in general and in database components in particular. Of course this warning applies to schema components as well. Presupposing that Glass' "facts" are true software reuse appears to be mainly a problem of software economy. The infrastructure specified above can be used for establishing an intra organizational market for database component at which a price for using or possessing components could be built. Vendors of components could improve their economical- and thus working

conditions based on the royalty they earn from others using their components.

7 CONCLUSIONS

In this paper it was supposed to consider the component concept for databases. It was indicated by listing examples of commercially available database components and by an example illustrating conditions in large data driven organizations that in fact a concept of database component may be useful. Basic database related concepts and in particular the concept database component were then defined. An infrastructure for dealing with database components and a usage model was then sketched. Finally potentials for reusing database components were shortly discussed.

REFERENCES

[BCN92] Carlo Batini, Stefano Ceri, and Shamkant Navathe. *Conceptual Database Design*. The Benjamin/Cummings Publishing Company; Inc., Redwood City, California, 1992.

[Bro97] Manfred Broy. Compsitional refinement of interactive systems. *Journal of the ACM*, 44(6):850 – 891, 1997.

[Che76] Peter P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–37, 1976.

[EN94] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, California et al., 1994.

[Gla03] Robert M. Glass. *Facts and fallacies of software engineering*. Addison - Wesley, Boston et al., 2003.

[Gra98] Ian Graham. *Requirements Engineering and Rapid Development*. Addison-Wesley Longman Limited, Harlow, England, 1998.

[HS00] Peter Herzum and Oliver Sims. *Business component factory: A comprehensive overview of component-based development for theenterprise*. John Wiley & Sons, Inc., New York et al., 2000.

[Kün04] Philipp Künsch. Ein Accounting- und Reporting - System für die Zukunft. *InfoWeek.ch*, 19:37 – 40, 25. Oktober 2004.

[Neu04] Hans Neukom. Early use of computers in Swiss banks. *IEEE Annals of the History of Computing*, 26(3):50 – 59, July - September 2004.

[Tha03] Bernhard Thalheim. Database component ware. In Xiaofang Zhou and Klaus-Dieter Schewe, editors, *Fourteenth Australian database conference (ADC 2003)*. Australian Computer Society, Inc., 2003.

[Wie03] R. J. Wieringa. *Design methods for reactive systems: Yourdon, Statemate, and the UML*. Morgan Kaufman Publishers, Amsterdam et al., 2003.

Appendix

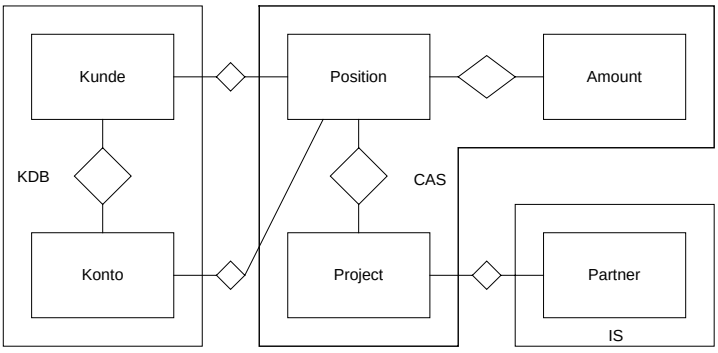

Figure 1 THE STRUCTURE OF DATA AVAILABLE TO CFS

Employee, as of 1 / 1 / 2003

(n1, STRING)	(n2, STRING)	(DOB, DATE)	((p, STRING), Project, leads)	((q, STRING), Project, works)
John	Smith	1 / 1 / 1984		red
Jane	Jones	20 / 5 / 1985	red	
Jim	Brown	12 / 3 / 1978	blue	

Project,as of 1 / 1 / 2003

(ID, STRING)	(start, STRING)	(budget, currency)
red	1 / 1 / 2003	1,000,000
blue	1 / 1 / 2002	0

Figure 2 AN EXAMPLE DATABASE