INFORMATICA, 2002, Vol. 13, No. 1, 47–72

Extraction of Object-oriented Schemas from Existing Relational Databases: a Form-driven Approach

Mimoun MALKI, André FLORY, Mustapha Kamel RAHMOUNI
Computer Science Department, University of Sidi Bel-Abbes, Algeria
LISI Insa-Lyon, 29 avenue Albert Einstein Villeurbanne, France
Computer Science Department, University of Oran, Algeria
e-mail: malki_m@yahoo.com, flory@insa.insa-lyon.fr, rahmouni@mail.univ-oran.dz

Received: May 2001

Abstract. In this paper, we present our Form-driven approach for reverse engineering of relationa databases. This methodology uses the information extracted from both form structure and instances as a database reverse engineering input using an interaction with a user. Through a combination of forms structures and data instances analysis, forms relational sub-schemas and their constraints are derived. These relational sub-schemas are mapped to object sub-schemas, which will be merging into global object-oriented schema that presents the whole underlying databases. The resulting global object-oriented schema must be validated as a rich and correct representation of the application domain.

Key words: relational databases, reverse engineering, object-oriented databases, forms, integration, validation.

1. Introduction

In many organizations, there are a large number of database applications that have evolved over many years. These systems are referred to as legacy systems (Farhrner, 1995a; Chiang, 1994; Hainaut, 1995; Soutou, 1998) and are characterized by lack of documentation and non-uniformity resulting from numerous extensions. These properties lead to inflexible systems and high maintenance costs.

Reverse engineering can help by understanding legacy databases and extracting their design specification (domain semantics). It is the part of database maintenance work that procures a sufficient understanding of a legacy database, and its application domain to allow appropriate changes to be made. More specifically, databases reverse engineering can be treated as a process that obtains domain semantics about a legacy database, makes the reverse schema transformation (from logical to conceptual) and, finally, represents the result usually as a conceptual schema (or object-oriented schema) (Farhrner, 1995b; Behm, 2000).

Users in organizations are quite experienced in the handling and manipulation of databases through forms. The CODASYL End User Facilities Committee (EUFC) has

recognized the importance of forms as the most natural interface between users and data and recommends the form-oriented approach as the main vehicle for user interface (Lefkovitz, 1979). Following these recommendations, there have been a number of automatic tools for defining, specifying, and implementing form applications (Batini, 1984; Shu, 1985; Yao, 1984; Choobinen, 1992); query languages and database systems have also been extended to deal with forms and other types of documents used in the office environment to facilitate the manipulation of data via computerized form (e.g., Oracle SQL Forms, Informix-SQL, Microsoft Access Forms and reports tools).

The objective of this research is to propose a form-based approach for reverse engineering of relational databases. By form, we mean any structured collection of variables (form fields) which are appropriately formatted to communicate with the databases especially for data retrieval and data display.

This approach uses the information extracted from both form structure and instances as a databases reverse engineering input using an interaction with a user. This proposition can be supported by the following arguments:

– a form model is a data model. Studying and analyzing a form and its relationship to other forms can reveal many data dependencies and mapping;

– usually the most widely used data are gathered or reported in a form and, therefore, important information can be obtained by analyzing a database's forms;

– forms are familiar and commonplaces objects, they are easy to read and understand;

– normally the forms are accompanied by instructions. The instructions provide additional information about organization's data and their procedural (or behavior) parts.

Our methodology articulate around five phases (Fig. 1):

– acquisition of the forms structure and instances,

– extraction of the domain semantics from forms analysis in order to construct their relational sub-schemas,

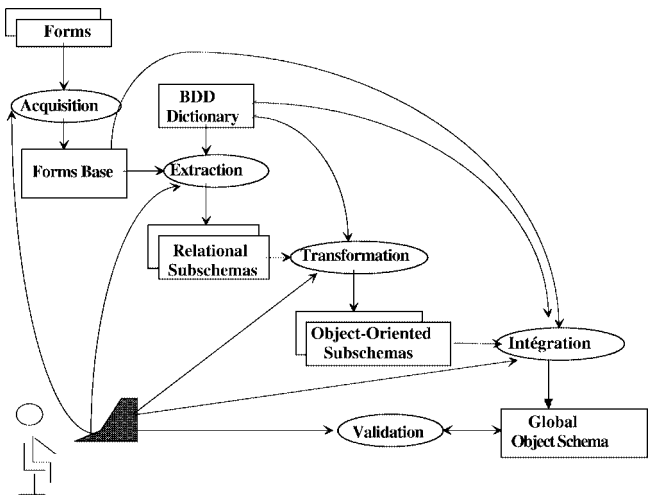

Fig. 1. Phases of form-driven approach of relational databases reverse enginnering.

– transformation of relational sub-schemas in object-oriented sub-schemas,

– integration of these object-oriented sub-schemas into a global object-oriented schema,

– validation of the resulting global object-oriented schema.

The paper is organized as follows. In Section 2, related approaches in database reverse engineering, are discussed. Section 3 contains the different stages of form acquisition while introducing the concepts of their model and their graphic description language. Section 4 presents the extraction of domain semantics from both form structure and instances, and construction of their relational sub-schemas. Section 5 describes the rules relational sub-schemas transformation in object-oriented sub-schemas, the integration of these sub-schemas into a global object schema and the validation of the resulting global object schema. Section 6 concludes the paper and summarizes the results.

2. Related Works

Several researches have been done on relational database reverse engineering, suggesting methods and rules for reverse schema transformation of the logical data schema of a legacy database into a conceptual (or object-oriented) schema. We classify these contributions in two categories.

In the first category, the extraction phase of domain semantics is not present. These approaches deal with the schema transformation, and have very strong requirements for legacy databases to be reversing engineered. For example, their process requires that the relations are in at least third normal form (3NF), a consistent naming convention is applied to attributes, and information on the availability of inclusion dependencies and keys is available. Farhner and Vossen (Farhrner, 1995a) classify these methods in two classes:

– Approaches, in the first class, elicit the semantics of the given relational schema through an evaluation of its inclusion dependencies (noted inds). Each ind is interpreted with respect to the role (key, part of key, foreign key, non-key) of its attributes. Most of these approaches require a complete set of inds and keys of the relations; some take only key-based inds into account (Casanova, 1983; Ji, 1991; Markowitz, 1990), while others consider general inds (Johanneson, 1994; Kalman, 1991).

– Approaches, in the second class, derive a conceptual schema through an evaluation of keys, their construction through other keys, and a discovery of foreign keys. Several of these approaches (Navathe, 1987; Batini, 1992; Chiang, 1994; Tari, 1997); also classify the relation of the source schema with respect to the construction of their keys, e.g., whether the primary key of a relation is the concatenation of the primary keys of other relations. The transformation is then done on the basis of this classification.

In the second category, the extraction phase of domain semantics is considered, such as keys, functional and inclusion dependencies, and integrity constraints by using machine-analyzable sources, such as databases instances and queries (or application programs). For example, a legacy database for reverse engineering normally contains a large

volume of data from application domain, so these data instances can be sources for extracting domain semantics.

– Approaches using application programs are based on the use of query language statements to extract (some of) the semantic information stored in a relational database (e.g., Anderson, 1994; Petit, 1995). Anderson (1994) analyses equi-joins statements whereas. Petit (1995) analyses auto-joins, set operations and where-by clause statements as well as the equi-join statements. Otherwise, the Hainaut's approach (Hainaut, 1995) analyses the code of programs, mainly the logical description of files in Cobol, for completing knowledge on the data structure and on the constraints.

– Approaches using extension of legacy database are based on the analysis of data instances to understand (some of) the semantics of database application. Chiang's approach (Chiang, 1994) uses the relational schema information as the basic input and aim to extract the semantic information by exhaustive analysis of each relation in the schema and their key and non-key attributes. Also Anderson (1994) and Permerlani *et al.* (1994) analyze data instances to extract inds from indication provided by joins. Otherwise, in (Tari, 1997), the classification of the relational schema is based on the analysis of data instances. Independently, Mannila *et al.* (1994) propose some algorithms (based-heuristics) to extract functional and inclusion dependencies.

Until recently, researches in database reverse engineering do not use forms, which manipulate databases, except Mfourga's approach, as machine-analyzable source. In (Mfourga, 1997), Mfourga proposes framework for extracting an entity-relationship model from a set of form model schemas of a legacy relational database. Form model schemas gather information, i.e., structural information and constraints among data, extracted from both their structures and instances. This approach uses these schemas to supplement low-level schemas schemes to help reconstruct a entity-relationship schemas.

Our approach emphasizes the extraction of the relational sub-schema of form, from both the form structure and the physical schema of the operational database, that is in fact a pivot schema reflecting the semantics of the operational database as well as the semantics of forms. So this schema facilitates the extraction of dependencies in order to construct the conceptual object schema of the underlying operational database.

3. Acquisition of Forms

This phase supports the description of forms and their instances on one hand, and, the automatic extraction of the forms hierarchical structures in order to facilitate their analysis and interpretation on the other hand. In this way, we use our form model presented hereafter.

3.1. *Form Model*

In (Malki, 1999), we have presented a general model of forms suitable for databases reverse engineering task. The model allows abstracting any database form, that is, to make explicit its components, fields as well as objects, and their interrelationships. It places a

special emphasis on finding relevant constraints among database form component by exploiting the form instances. This model is similar but not identical to the models presented in (Choobineh, 1992; Mfourga, 1997).

3.1.1. *Form Type*

A form is a structured collection of variables (i.e., fields) appropriately formatted to communicate with a database. A form type defines the structure, constraints and presentation of the form fields. It represents the form intension as perceived by users. A particular representation of form type is called a form template. A form template not only specifies a representation, but is also medium dependent. This paper only addresses screen and paper forms, which are used as communication interfaces with the database.

Three basic components of any template are title, captions, and entries. Every form must have a title, which names the form and provides a general description of what is to be filled in by the respondent. The title of the form in Fig. 2 is "CARD OF WAGE". Captions are preprinted on the form. They serve as a clue as what is to be filled in by the respondent as well as a guide to enter or to read it on the form. An entry is the actual data that is entered by an operator or displayed by the form processing system on the space provided. The entry data may be text, numeric, bitmap or any other user defined data type. As an example in Fig. 2, Name is the caption associated with the entry with the value of "Malki".

A form instance is a particular collection of values for the form fields. When a form template is filled with values, it becomes an instance of that form type. Fig. 2 is a template of "CARD OF WAGE" form with the values of a particular form instances.

3.1.2. *Structural Units*

Three types of form fields can be identified: atomic, box, and multiple choice (table). An atomic field does not have any constituent (sub) fields. A box (or table) field groups a

University of Djillali Liabes
22000 Sidi Bel abbes

CARD OF WAGE

Month: October 2000

Employeeid	Name-Emp.	Birth-date	Ms	Child	Ssn	Mutualn	Accountid
7850012A	MALKI	12/01/1959	m	2	14785	147522E	7814012B2

Function	Department	Rank	Design.	Cat.	Sect.	Level	Seniority
Teacher	Computer	g01	Professor	D1	05	07	09-22-1986

PREMIUM & INDEMNITIES				DEDUCTION			
Id	Designation	Rate	Amount	Id	Designation	Rate	Amount
e01	Base wage		25000	r01	tax		5000
p01	Pedagogy		9500	r02	ss Contributions		2500
a01	Fam. allow.		900				

TOTAL AMOUNT	DEDUCTION AMOUNT	NET AMOUNT
35400	3000	32400

Fig. 2. Example of form: "Card of wage".

number of other fields. A form field roughly corresponds to the concept of an attribute in database terminology.

A structural unit type (SU), is a box (or table) of homogeneous pieces of information, that is, an object that groups closely related form fields. Each SU is a logical sub-part of a form type. The Fig. 2 shows some structural units of the "CARD OF WAGE" form: Employee Identification, Function/activity and Premium/indemnities.

3.1.3. *Relationships between Structural Units*

We use the parent-child relationship which is one-to-many (multi-valued) or one-to-one (single) relationship between two structural units. One of the SUs (always the one-side) is called the parent SU, the other (many-side or sometimes one-side) is called the child SU. An occurrence of a relationship consists of one SU occurrence of the parent and one or several occurrences of the child SU. In the Fig. 2, the relationship between the SU "CARD OF WAGE" (parent) and the SU "Identification" (Child) is one-to-one, i.e., that a "CARD OF WAGE" is established for only one Employee. On the other hand the relationship between the SU "CARD OF WAGE" (parent) and the SU "Premiums/indemnities" is one-to-many, i.e., that an employee may receive a number of premiums and indemnities.

In this way, our model permits a hierarchical structuring of form fields. This hierarchical structure corresponds to form type logical structure. The root of the tree is the form title. The other nodes represent structural units.

Children that have the same parent belong to the same level. The root level is number 0 and each subsequent level is one greater than its parent's. The Fig. 4 shows the hierarchical structure of the form "CARD OF WAGE".

One may note that the form model includes a logical and a physical aspect.

The logical aspect corresponds to the form definition, as it is perceived by users (or designers), i.e., its intensional part. This intensional part is a logical structure that contains logical sub-parts corresponding to the objects (structural units) that compose a form viewed as a parent-child hierarchical structure. The logical aspect allows understanding both the form structure and meaning.

The physical aspect of a form is represented by a given form template. In this template, the title of the form provides a general description of what is to be filled in. The placement of the fields is often well studied and homogeneous information is in general contiguous. Field captions are in general more explicit than underlying database attributes. The template also provides instructions that represent contextual knowledge that is knowledge about objets represented on the forms. Therefore, the form physical aspect can be useful for identifying the objects contained in a form.

Now, we can define the notion of the form base representing a legacy (or operational) database application. This base possesses for every form: its description, its hierarchical structure, and its instances. Formally, a form base is a triplet $B(D, S, I)$, where D represents a form description (i.e., both physical and logical aspects), S represents a hierarchical structure of form and I represents their instances.

3.1.4. *Graphic Language of Form Description*

In (Malki, 1999), we have presented a graphic language of form description (noted GLFD). This language, that is a graphic interface, assists the user (or designer) in specifying form description from concepts of the form model presented above. The idea consists in identifying fields and structural units of the form (logical aspect) from its graphic representation or form template (physical aspect).

We distinguish two definition shapes of the GLFD (graphic and declarative) that are jointly usable.

3.1.4.1. *Graphic Definition.* It would be helpful to use the graphic definition of the GLFD in order to facilitate the form description through its graphic interface while using the base concepts of the form model: group (or box), list (or table) and fields. This graphic interface describes physical aspect of form while considering the structural units as box (or table) composed with text, graphic and information (field). This identification is sometimes enforced by separators (e.g., blanks, lines).

3.1.4.2. *Declarative Definition.* The base concepts of the form model used by the graphic interface are expressed also under declarative specifications. These declarative specifications describe the features characterizing every element (field, and structural unit) of form reproduced graphically.

The syntax of this declarative specification is the following:

Box: <Name-Box>
Identify: <identify-Box>
Composition :{ <Identification-field>, <Text-Block>,
<Graphic-Block>,<Identify-Box>}
End-Box
Table :<Name-table>
Identify : <Identify-table>
column: $1/N$
Line: $1/M$
Field-Column or Field-line: $< N^{\circ} -$field> {<name-field>}
End-table
Field : <Name-field>
Identify : <Identify-field>
Type: integer/real/Boolean/character
Length: <integer>
Origin: System/User/computing <computing-rules>
level : <integer>
End-field

Let us take an example of the list specification “premium & indemnities ” of “CARD OF WAGE” (see Fig. 3).

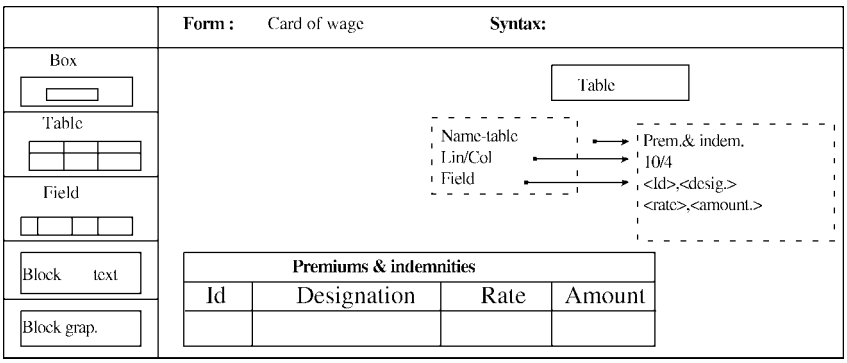

Fig. 3. Example of specifation of table acquisition.

3.2. *Insertion of Forms*

In this step, we use a graphic language based on the form model concepts presented above, for describing the form according both to their physical and logical aspects. In the same way, a form dictionary is used in interaction with the designer. This dictionary, which will be integrated in the form base, contains the meaning of fields (designation, synonym, and rules of computed field). This dictionary is used to insure a presentation as complete as possible for the form analysis.

For each form, we take several instances. These instances will be used with the formal description of form as machine-analyzable source in order to facilitate the extraction of the domain semantics.

3.3. *Structuring of Forms*

This process, that explores both physical and logical aspects of forms described by the graphic language, identifies the structural components of form and their interrelationships in order to construct the hierarchical structure of the form. It would be helpful to have a precise picture of the hierarchical relationships of a form in order to clearly understand its meaning and facilitate the extraction of the domain semantics.

This procedure, which is an automatic process and transparent to the designer, constructs the hierarchical structure in four steps:

– defining the root node whose name is the form's title. All fields are reattached to the root node;

– transforming in the first all structural units in node with level 1, i.e., their parent is the root node;

– identifying the parent-child relationships among structural units. Because there are several level of fields (e.g., 0, 1, 2, 3, etc.), this process identify recursively the sub-structure (or sub-node) according to levels of their fields specified in the insertion process of forms;

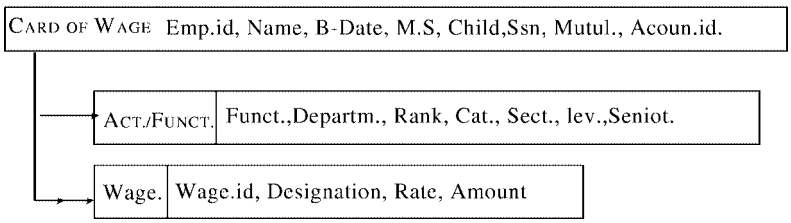

Fig. 4. Hierarchical structure of the form "Card of Wage".

– and simplifying the hierarchical structure while eliminating sub-structures that contain only the text (or graphic) and calculated fields that do not contain interesting information.

For example, the form "CARD OF WAGE" is restructured in hierarchical structure (see Fig. 4).

4. Extraction of the Domain Semantics

Forms are often the most convenient interfaces for entering, changing, and viewing data in database applications. Therefore, the most widely used data are gathered or reported in forms. They are designed mostly with a deliberate attempt at user-friendliness and they do not necessarily directly mirror the structure of the underlying database.

A deliberate effort of abstraction (i.e., description and structuring of forms: see section above) is necessarily for making all the components of the form individually accessible, and explicating their interrelationships as well as the constraints (or dependencies) on the data they display. In this way, the forms of a given database taken as a whole can be used to recover database semantics, especially with legacy systems. Indeed, with such older systems, database applications often do not come with a conceptual schema, but only, for example, with a relational schema.

In the relational model, semantics are specified not in the structure, but through constraints. Data dependencies are one kind of these constraints; where functional and inclusion dependencies are of particular interest. Nowadays, relational databases are spread everywhere, and many applications like reverse engineering, integrated access for interoperable databases, knowledge based interfaces, migration to newer database management systems (DBMS), and improvement on the effective utilization of the data of the organization require a deep knowledge on the semantics of these existing databases. Unfortunately, many relational DBMS do not provide much support for specifying data dependencies, and consequently, this semantics cannot be found in the schema. However, since data dependencies are implicit in the extension of the databases, a solution to this problem is to extract the dependencies by analyzing the extension.

The goal of this phase of extraction is to derive the relational sub-schema of form from their hierarchical structure and their instances according the physical schema of the underlying database. This process, in the first, identify relations and their primary

keys respectively with regard to both structural units (or nodes) of form and underlying database, then extract the functional and inclusion dependencies through both their hierarchical structure and instances.

Since the forms group real objects (or structure units) which are manipulated by end-user, this process of extraction does not necessarily restructuring the relational subschemes representing these forms on the one hand, and facilitates the derivation of relations as well as the identification of objects (or structure units) on the other hand. Otherwise, it is not often true, when one takes only the physical scheme of database as machine-analyzable source (Chiang, 1994; Petit, 1995). In addition, the using of form instances reduces the time of extraction of domain semantics, with regard to approaches that use database instances as machine-analyzable source (Chiang, 1994; Mannila, 1994; Petit, 1995).

We start briefly by introducing the concepts of the relational model (Ullman, 1984).

4.1. *Relational Model*

In this sub-section, the basic relational concepts and terminology used in this paper are reviewed. Details can be found in any textbook on the relational model, e.g., (Ullman, 1984; Elmasri, 1994).

A relation scheme is a pair <R_i, X_i>, usually denoted $R_i(X_i)$, where R_i is a relation scheme name, and X_i denotes a sequence of attributes. Each attribute of a relation scheme is associated with a set, which is called its domain. We use the following notational convention: if $R_i(X_i)$ is a relation scheme with m attributes, we write $X_i = A_{i,1}, A_{i,2}, \ldots, A_{i,m}$. The domain for $A_{i,j}$ is denoted by $D_{i,j}$. A tuple for the relation scheme $R_i(X_i)$ is an element of $D_{i,1} \times \ldots \times D_{i,m}$.

A relation for a relation scheme $R_i(X_i)$ is a set of tuples for $R_i(X_i)$. If t is a tuple for $R_i(X_i)$ and Y is a subset of X_i, then $t[Y]$ denotes the subtuple of t corresponding to Y.

Let $R_i(X_i)$ be a relation scheme, r_i a relation for $R_i(X_i)$, and Y a subset of X_i. The projection of r_i on Y, denoted by $\pi Y(r_i)$, is $\{t[Y]/t \in r_i\}$.

A functional dependency (noted fd) over R_i is a statement of the form R_i: $Y \rightarrow Z$, where Y and Z are subsets of X_i. A functional dependency Ri: $Y \rightarrow Z$ is satisfied by r_i if for any two tuples, t and t', in r_i, $t[Y] = t'[Y]$ implies $t[Z] = t'[Z]$.

A key for R_i is a subset K_i of X_i, such that R_i: $K_i \rightarrow X_i$ is satisfied by any r_i associated with R_i and there does not exist any proper subset of K_i having this propriety. A relation scheme can have several candidates keys from which one primary key is chosen. Therefore, in a relation, attributes can be classified into primary key, foreign keys, which are keys in an other relation and non-key attributes. A primary key (or a sub-part of a key) can be at the same time a foreign key.

In any algorithm that discovers fds from extension of the relations, it is important to minimize redundant work, that is, the work of analyzing the tuples to obtain redundant fds, because they can be derived from other ones by applying Armstrong rules (Armstrong, 1974). The focus is on the following derived fds:

1. Trivial fds rule: $X \rightarrow Y$ if $Y \subseteq X$.

2. Augmentation rule: $X \rightarrow Y$ then $XW \rightarrow YZ$, if $Z \subseteq W$.

3. Transitivity rule: if $X \rightarrow Y$ and $Y \rightarrow Z$, then, $X \rightarrow Z$.

The presence of a fd can give rise to redundant information, which can be eliminated by performing a decomposition. This observation has led to the definition of several normal forms of relational database (1NF, 2NF, 3NF, 3NFBCK). In our approach, we suppose that relations are in 3NF.

Let $R_i(X_i)$ and $R_j(X_j)$ be two relations schemas. Let r_i and r_j be relations of $R_i(X_i)$ and $R_j(X_j)$, respectively. An inclusion dependency (noted ind) is a statement of the form $R_i.Y << R_j.Z$, where Y and Z are ordered subsets of X_i and X_j, respectively. The inclusion dependency $R_i.Y << R_j.Z$ is satisfied by r_i and r_j if $\pi Y(r_i) \subset \pi Z(rj)$. An inclusion dependency is key based if its right hand side is a key.

A relational schema is a pair $< R, D >$, where R is a set of relation schemes and D is a set of functional and inclusion dependencies.

Let us take an example of relational schema of "Personnel" database from which we apply our reverse engineering approach while analyzing their forms:

Employee (**emp.id.**, name, birth-date, address, sex, m.s., number-chil., ssn,
accountid, nationality, recruitment-date),
Child (**emp.id., name**, birth-date, sex)
Holiday (**hol.id**, designation)
On-holiday (**hol.id, emp.id**, begin-date, end-date)
Element-Paye (**payeid**, designation, rate-number, type)
Payed (**payeid, emp.id**, date)
Sanction (**sanc.id**, désigantion, degree)
Punished (**sanc.id. emp.id**, sanction-date)
Diploma (**dipl.id**, designation)
Graduate (**dipl.id, emp.id.**, obtain-date, University)
Rank (**rankid**, designation, category, section, level)
Having-rank (**rankid, emp.id**, begin-date, end-date)
Function (**func.id**, designation)
Having-function (**func.id,emp.id,dept.id**, begin-date, end-date)
Departure (**dep.id**, designation)
On-Departure (**dep.id,emp.id**, date, length)
Department (**dept.id**, designation, emp.id), Bank (**bankid**, name, address.)

4.2. *Identification of Form Relations*

The forms either permit the updating of relations in underlying database or represent a view that is a joint of relations. Therefore, each field entry is generally linked to an attribute of one relation in the underlying database. However, the identification of form relations and their primary keys respectively, consists in determining the equivalence and/or the similarity between structural units (nodes) of hierarchical structure and relations in the underlying database. This is a basis point from a reverse engineering point of view (Malki, 1999).

A node of a form hierarchical structure may be either:

– equivalent to a relation in the underlying database, i.e., these two objects (node and relation) have a same set of attributes;

– similar to a relation, i.e., its set of attributes is a subset of the one of the relation;

– a set of relations, i.e., its set of attributes regroups several relations in underlying database.

Also, for dependent nodes (or form relation), primary keys are formed by concatenating the primary key of its parent whit its local primary key.

This process of identification is semi-automated because it requires the interaction with the analyst to identify objets that do not verify proprieties of equivalence and similarity.

While applying this process on the hierarchical structure of "Card of wage" and the physical relational schema of underlying database, we extract the following relational sub-schema:

Employee (**emp.id.**, name, birth-date, address, sex, m.s., number-chil., ssn,
accountid, nationality, recruitment-date),
Payed (**payeid, emp.id**, date)
Having-rank (**rankid, emp.id**, begin-date, end-date)
Having-function (**func.id, emp.id**, dept.id, begin-date, end-date)

4.3. *Extraction of the Functional Dependencies*

The extraction of functional dependencies from the extension of database has received a great deal of attention (see, e.g., Anderson, 1994; Castellanos, 1993; Mannila, 1994; Petit, 1995). The aim of the extraction of functional dependencies is to reduce the complexity of extraction algorithms, that is exponential time with regard to the number of attributes (left hand side of fd noted lhs), and polynomial with regard to the number of tuples of relation in database (right hand side of fd noted rhs).

In our approach (Malki, 1999), we introduce two ways to reduce the time for extracting functional dependencies. The first one replaces database instances with a more compact representation that is, the form instances. The second one is to use only the non-key attributes as left hand side of functional dependencies, because, we suppose that the relational schema is in the 3NF (where the primary key determines functionally the rest of attributes, i.e., non-key attributes). In this condition, it is possible to infer a new fd through non-key attributes and key attributes (i.e., non-key attributes $\rightarrow$ key attributes). However, we consider the utilization of forms as sampling database for extracting domain semantics.

Let L_a be the left-hand side (simple or composite) of a fd, that is, the determinant, and R_a the right hand side (always simple). For every pair (L_a, R_a) all the tuples of the relation must be analyzed to test the fd condition.

Now, we present the fundamental principles that permit to optimize the process for extracting the fds:

– The possible L_a's are ordered by number of attributes (i.e., non-key attributes). For a relation R with non-key attributes A, B, C, the order would be: A, B, C, AB, AC, BC,

ABC. This order guarantees that no L_a is analyzed until any subset of its component attributes has been analyzed.

– For every possible L_a X that is analyzed, each key attribute of R is candidate as a R_a of a possible Fd. However, each such attribute is proposed as a possible R_a only if there is no previously found fd whose left hand side is contained by this L_a and whose right hand side is equal to this R_a, otherwise, the pair (L_a, R_a) represents a redundant fd (by augmentation rule) and R_a is not a possible but a positive right hand side for La. In this way, only fds with minimal L_a's are obtained. Furthermore, once a fd has been discovered, all transitive fds that can be derived at that moment are computed. With this mechanism, the analysis of the extension of the relation (sub-relation of form) is avoided for many redundant fds.

– For each pair (L_a, R_a) representing a possible fd, the extension of the relation is analyzed to test if it satisfies the Fd condition. This condition is checked simultaneously for all possible fds (i.e., all possible R_a) on a L_a while the pairs of tuples ti, tj are being retrieved. The problem is that to test the fd condition for each different value b for L_a, every tuple in the relation must be retrieved in search of the same value b for the condition of attributes composing L_a. If the search succeeds, then for each possible R_a the corresponding attributes of both tuples are checked to see if their values match too.

– The sorting of the relations according L_a reduces the search of R_a.

The algorithm for extracting fds through sub-relation form is presented in (Malki, 2000), only the main parts are given below.

Algorithm 1. Extraction of the functional dependencies

Input:
RS: Relational sub-schema $< R'(Y), K'$: primary key>
FB: Forms base $< T$: template, S: hierarchy structure, I: instance of $R'_i >$
Output:
FD: Set of fds: $L_a \rightarrow R_a$
/L_a: attributes no keys and R_a s: key attributes/
$FD := \emptyset$
For each Relation R'_i
$L_a := X_i - K'_i$ /the set of non-keys attributes/
For each La_k
$Comp - Ra(X - La_k)$
/it is key attribute under atomic shape for which $La \rightarrow Ra$ is no redundant/
If Ra then
Sort $r'_i(La_k)$
Detect-fd (La_k, Ra, r'_i, FD) /Test this fd. according to instances of form/
Endif
Transit-fd (Fd) /deduct all fds. while applying the rule of transitivity/
Endfor
Endfor
FAlgo

Detect-fd (La, Ra, r, FD)
$i := 1$
$n := card(r)$
While $Ra \neq \emptyset$ or $i < n$
If $t_i[La] = t_{i+1}[La]$
For each Ra_j
If $ti[Ra_j] \neq t_{i+1}[Ra_j]$
$Ra := Ra - Ra_j$
Endif
Endfor
Endif
$i := i + 1$
Endwhile
If $Ra \neq \emptyset$
$FD := FD + (La, Ra)$
Endif
Endetect

While using non-key attributes, the number of candidates to extract a canonical functional dependence becomes therefore linear with regard to the number of attributes. The time complexity is then reduced to $\bigcirc\,(n\,|X|\,p \ln p)$, where $n = |X_i|$, X is a the non-key attribute and P is number of tuples in r_i.

While applying this algorithm on the sub-schema of "Card of Wage" and their instances, one finds the Fd: departmentid → Employeeid. These functional dependencies, that are extracted, can be used for normalizing correspondents' relational sub-schemas in 3BKCNF.

4.4. *Extraction of Inclusion Dependencies*

Once the extraction of the functional dependencies is achieved, one proceeds to the extraction of the inclusion dependencies, which contain the majority of the information for the identification of relationships between relations. The problem with the extraction of Inds is well known (Chiang, 1994; Castellanos, 1993; Mannila, 1994; Petit, 1995). The naive method of checking, for each part of relation with n attributes in the database, all possible Inds, one by one, is particularly impossible, because there are $n!$ possibles no equivalent Inds. In order to reduce the set of possible Inds, Chiang *et al.* in (Chiang, 1994) and Petit *et al.* in (Petit, 1995) use heuristics that only formulate Inds between relations' key attributes. These possible inds will be verified by analyzing data instances in regard with Ind definition.

In our approach (Malki, 1999), we formulate possible Inds between relations' key of relational sub-schema of form. The time of this process is more optimized with regard to the other approaches ((Chiang, 1994) and (Petit, 1995)), because the possible Inds are verified by analyzing the form extensions which are more compact representation with regard to the database extension.

The heuristics employed by the extraction process for proposing possible inclusion dependencies are described in below.

– Two relations R_i and R_j, having the same primary key X, may be linked by a Ind: $R_i.X << R_j.X$,

– When the primary key X of a relation R_i is a foreign key x in an other relation R_j, then it is possible to have a Ind: $R_j.x << R_i.X$.

In addition, we apply some criteria, which verify the Inds proposed by analyzing data instances of forms:

– For each formulated Ind, the extraction process must compare the sets of values appearing in $R_i.X$ and $R_j.X$ according to the Ind definition,

– Redundant Inds can be detected by the inference rules of inclusion dependencies (projection and transitivity) (Elmasri,1994), that is: If $R_i.X << R_j.X$ and $R_j.Y << R_k.Y$ hold, and Y is a subset of X, then the Ind: $R_i.X << R_k.Y$ is redundant.

The algorithm of extraction of Ind is detailed in (Malki, 2000); we present hereafter only the main parts.

Algorithm 2. Extraction of Inclusion Dependencies

Input:
RS: Relational sub-schema: $< R'(X), K'$: primary key>
SRS: Set of RS of Forms
FB: Forms base: $< T$: template, S: hierarchy structure, I: instance of $R'_i >$
Output:
IND.: Set of the Inds: $R'_i.X_i << R'_j.Y_j$
Begin
IND:= $\emptyset$
For each R'_i of SRS
For each $R'_j \neq R'i$ of SRS
/Test of a possible Ind. joining two relations even having the key primary/
If $K'_i = K'_j$
Test-Ind $(R'_j.K'_i, R'_i.K'_i, IND)$ /According to Instance of form/
Else If K'_i =Foreign key(R'_j)
Test-Ind $(R'_j.K'_i, R'_i.K'_i, IND)$
EndIf
EndForj
EndFori
Endalgo
Test-Ind $(R_i.X, R_j.Y, IND)$
If $(R_i.X, R_j.Y) \notin IND$ /it is a Ind. no redundant/
$r_i.X := \pi X(r_i)$ /The projection of r_i on X/
$r_j.Y := \pi Y(r_j)$ /The projection of r_j on Y/
If $r_j.Y \neq \emptyset$ and $r_j.Y \sqsubseteq r_i.X$
$IND := IND + (R_j.Y << R_i.X)$
EndIf

If $r_i.X \neq \emptyset$ and $r_i.X \sqsubseteq r_j.Y$
$IND := IND + (r_i.X << r_j.Y)$
EndIf
Transit-Ind (IND) /deduct all Inds. From the rule of transitivity/
EndIf
EndTest-Ind

In this algorithm, attributes of dependencies are the primary keys and foreign keys. Thus, the time complexity is reduced to the test of the inclusion dependency on the form instances.

The set of the Inds extracted from the relational sub-schema "Card of wage" is:
Having-rank.**rankid** « Rank.**rankid**
Payed.**payeid** « Element-Paye.**payeid**
Having-function.**func.id** «function.**func.id**

5. Transformation of Relational Schema into Conceptual Object Schema

Object-oriented technology is being widely used in software development these days. The object-oriented data model is growing in popularity in the area of database development. Consequently, there is also a growing interest in re-engineering relational databases to object-oriented databases. Reverse engineering of relational databases is concerned with the task of extracting structures corresponding to a particular conceptual model such as the entity-relationship (ER) model, the extended entity-relationship (EER), the object-oriented (OO) model, and so on (Behm, 2000; Farhrner, 1995b; Ramannathan, 1997; Tari, 1997).

The fact that our approach uses database forms as machine-analyzable source, implies that we first transform the relational sub-schemas of forms into object-oriented sub-schemas. In the following, we merge the set of object-oriented sub-schemas in a global object-oriented schema representing the conceptual schema of the underlying database. The resulting global object-oriented schema must be validated as a rich and correct representation of the application domain. As a target model, we use the object-oriented model called HCB (Ayache, 1995) developed in our laboratory. It is presented hereafter.

5.1. *The HCB Model*

The HCB model is an object-oriented model with a three-dimensional representation of objects. HCB is first a model of representation, because it allows the specification of the object scheme, following three planes (Fig. 5a): structuring, communication and inheritance, where each component is described separately and represents the projection of the object on the corresponding plane. A simple representation of objects gives, in general, a diagram composed of heterogeneous information and different link types. The comprehension and the exploitation of such schema are not easy because of the superposition of the different perspectives. The HCB model aims first to offer an object schema, which is coherent and easily exploitable. The model is called HCB because each perspective

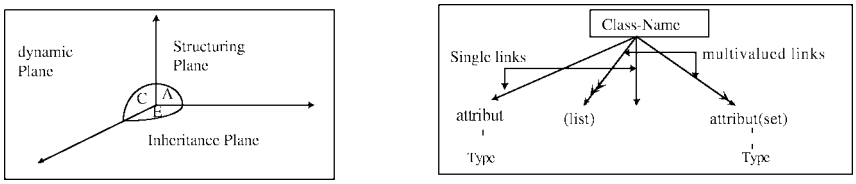

Fig. 5. a) The three planes of the model HCB; b) Representation of a class.

represents a specific verb. The structure is modeled by the verb Have because it specifies what the object has. The behavior is modeled by the verb Communicate. Finally, the inheritance perspective is modeled by the verb Be because it express what the object is.

5.1.1. *Constructors of the HCB Model*

The structuring of information is done using the following constructors to build composed object types (Fig. 5b.):

– tuple: if a class contains component of different types, those latter will be structured by the tuple constructors. The tuple is a structure of heterogeneous objects, where order is not significant. The structure can be recursive since a tuple may be used to construct another tuple.

– set: the set constructor is a mechanism allowing the creation of a "set-object" from homogeneous component objects of same type, where the order of object member is not significant.

– list: the list constructor allows the grouping together of homogeneous objects, where the order is significant and an element may be repeated.

5.1.2. *Composition, Aggregation and Inheritance*

The composition links introduce a semantic of cardinality between an object and its components (attributes). To keep this information, we propose the following representation:

– a single-valued link is represented by a simple arrow (noted $\rightarrow$),

– a multi-valued link is represented by a double arrow (noted $\rightarrow\rightarrow$), and we use a triple arrow when the values form a list (noted $\rightarrow\rightarrow\rightarrow$),

– a link of inheritance (i.e., link of specialization/generalization) is represented by a double links oriented from subclass to superclass,

– in addition, the concept of role allows the modeling of crossed references between entities of the system. The HCB model generalizes the notion of role and uses it any time that the attribute value is a reference to an object belonging to another class(e.g., Empid of department class: Fig. 6a).

The dynamic part of an object overlays two aspects, calculation (i.e., methods with communication messages) and behavior (i.e., life cycle). We distinguish two categories of methods:

– interfaces methods, which allow access to values of attributes. We can write, read, modify, and perform other access functions on attribute values;

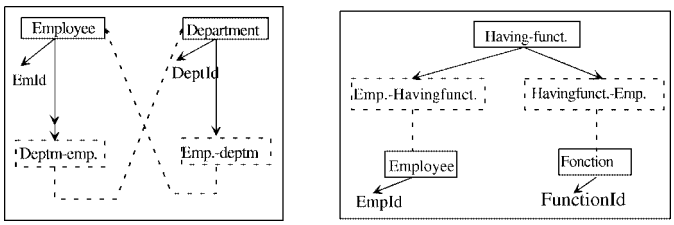

Fig. 6. a) Binary association; b) N-ary association.

– calculation methods allowing computation of attribute values. Calculation methods send messages to access instances.

5.2. *Transforming Form Relational Sub-Schema into Object-Oriented Sub-Schema*

Formally, a schema mapping of a source schema S_1^{M1} defined on a data model $M1$ to a target schema S_2^{M2} defined on a model $M2$ may be defined as a transformation T such that $T(S_1^{M1}) = S_2^{M2}$. The transformations are usually a collection of rules that map concepts from the source data model to the target data model (Hainaut, 1995; Ramannathan, 1997).

The proposed methodology of transformation, which is a mixture of key-based and Ind-based approaches (Farhrner, 1995a; Chiang, 1994; Petit, 1995), consists in identifying the components of the object sub-schema in the HCB model through the relational sub-schema. However, this process that concerns all three planes of HCB model (i.e., structuring, and inheritance) permits that:

– the identification of object classes and their composition relationships will be presented in the structuring plane;

– the identification of inheritance relationships will be presented in the inheritance plane;

– and the extraction of the dynamic aspect of object classes from the forms will be presented in the communication plane. This process will be investigated subsequently in other work.

5.2.1. *Identification of Object Classes*

In order to simplify the translation process, we assume that all relations (or relation scheme) are in the 3NF. The reason for this is to ensure that each relation corresponds to exactly an object class. These object classes have the same attributes as those contained in the relations. Thus, we define a class-relation to be a relation that directly maps to an object class.

5.2.2. *Identification of Associations*

In the HCB model, we identify two types of associations: binary and n-ary associations. These associations are represented by reference links (i.e., role attribute) and association classes respectively.

5.2.2.1. *Binary Associations.* The foreign keys of class-relation and the corresponding Inds identify a binary association between class-relations. Therefore, this referential link is translated in role attribute in the HCB model. The target will be, in general, a role attribute typed by the other class. As the association is not directed, then the obtained represented comprises crossed references, as shown in the following schema (Fig. 6a).

While applying this transformation rule on the two class-relations and their Inds below, we generate the following object schema (see Fig. 6a).

Employee (**emp.id.**, emp.name, birth-date, address, sex, m.s., number-children, ssn, accountid, nationality, recruitment-date),

Department (**deptid**, design, empid),

IND: Department.**empid** « Employee.**empid**.

5.2.2.2. *n-ary Associations.* Every class-relation whose primary key is entirely composed of foreign keys is a class that represents an association (noted association-class) between all the classes corresponding to the class-relation that foreign keys refer to. This n-ary association may be translated by an attribute tuple. In this transformation step, an option must be taken and the possibility is let to the analyst to choose one of the transformation rules. Although the creation of association-class transforming the n-ary association can become indispensable.

The relation Having-function (**funcid, empid**, deptid, beginning-date, end-date) is translated into Association-class as show in Fig. 6b.

5.2.3. *Identification of Inheritance Relationships*

Two forms of relational structures may indicate an inheritance relationship:

– In the first case, every pair of class-relations (CR_1, CR_2) that have the same primary key (noted X) and the corresponding Inds (i.e., $CR_1.X << CR_2.X$) may be involved in an inheritance relationship, i.e., CR_1 "is-a" CR_2. In Fig. 7a, the relations Employee and Child have the same primary key (empid) and the corresponding Ind: Child.empid « Employee.empid, therefore Employee is the superclass and Child is a subclass.

– In the second case, only one class-relation is considered to which an inheritance relationship may be generated. For example, in the relation Employee, the attribute rank is relevant only for the teacher employee (i.e., EmpType="T") and the attribute Function is relevant only for administrative employee (i.e., EmplType="A"). The relation contains

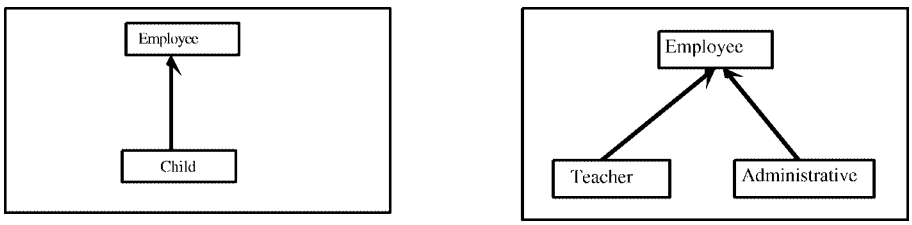

Fig. 7. a) on two relations b) on only one relation.

null value for rank attribute for all administrative employees and null values for Function attribute for all teacher employees. This is an example of instances of two subclasses being stored in the single relation.

The identification of this form of inheritance relation raises the knowledge domain from database (Piatetsky, 1991). There are several knowledge discovery algorithms that can identify rules from relational database (Piatetsky, 1991). By using one of those algorithms on the Employee relation, for instance, we can identify the following rules:

EmpType = "T" $\Longrightarrow$ Function = Null (1)

EmplType = "A" $\Longrightarrow$ Rank = Null (2)

Such types of rules are typically called "strong rules" since they hold for all the instances of the relation. In effect, these algorithms can be used to identify the discriminate attribute of the generalization by finding rules whose right hand side is of the form A =Null where A is some attribute name. Following is the generalized procedure for identifying sub-classes once the characteristic rules have been identified.

As an example, the rules ((1) and (2)) permit to identify the hierarchy of generalization with two sub-classes (see Fig. 7b).

Regarding the transformation rules presented above, we propose an algorithm that transforms a form relational sub-schema into an object-oriented sub-schema.

Algorithm 3. Transformation of a relational sub-schema into OO subschema

Input :

RS: Relational sub-schema: $< R'(X), K'$: primary key>

FB: Forms base: $< T$: template, S: hierarchy structure, I: instance of $R'_i >$

IND: Set of Inds.: $R'_i.X_i << R'_j.Y_j$

Output:

OS: Object-oriented sub-schema: $< C$: classes, L: composition, H: inheritance>

$C := \emptyset$; $L := \emptyset$; $H := \emptyset$;

For each R'_i relation

Create_class C_i

$C := C + C_i$ /each relation becomes a Class/

Attachment (X_i, C_i) /Attachment of all attributes of R'_i to C_i/

Endfor

For each R'_i relation

If $\exists\, Ind$: $R'_i.X_i << R'_j.Y_j$

If X_i is Primary Key of R'_i

$H := H + (C_i{-}\text{isa}{\rightarrow} C_j)$ /An inheritance of two class relations (C_i, C_j)/

Else If $\exists k'_i \in K'/X_i \subset k'_i$

If k'_i groups together a partition of n foreign key referencing for example the following relations: $R'_{i,1} \ldots R'_{i,n}$

create_association_class C_a

$C := C + C_a$ /Creation of an association class (i.e., Association n-ary)/

For each $R'_{i,l}$: C_l /Class participant to the association/

create_attribute T_l of C_a /typed Role by C_l/

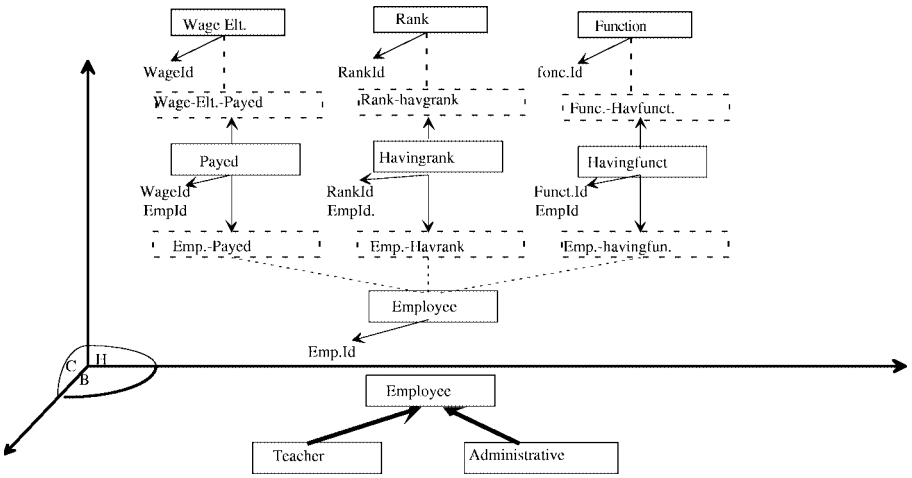

Fig. 8. Object-oriented sub-schema of "Card of wage".

$L := L + (C_a$ – role $\rightarrow C_l)$
Endfor
Else If $X_i \not\subseteq k'_i$
create_attribute T_i /typed Role C_j/
$L := L + (C_i$-role$\rightarrow C_j)$ /generation of a binary association (role)/
create_attribute T_j /typed Role C_i/
$L := L + (C_j$-role$\rightarrow C_i)$
Else Identify the strong rules: $(X_a = value) \Longrightarrow X_i$ =Null
create_super-class C_h whose a set of attributes $= X - \cup X_i$
For each X_i
create_subclass $C_{h,i}$
$H := H + (C_{h,i}$-isa$\rightarrow C_h)$ /An inheritance of only one Relation/
Endfor
Endif
Endfor
Endalgo

Fig. 8. presents the transformation of the relational sub-schema of "card of wage " into an object-oriented sub-schema.

5.3. *Integration of Object-Oriented Sub-Schemas*

In this phase of reverse engineering using forms as machine-analyzable source, object-oriented sub-schemas are derived from relational sub-schemas. These object sub-schemas will be merging into a global object-oriented schema that represents the whole underlying database. However, we apply the techniques of integration schema. We assume, in agreement with (Batini, 1986) that the integration schema process consists in two phases: comparison and conforming of schemas, and merging and restructuring of schemas.

The comparison phase performs a parities comparison of objects (of the sub-schemas) and finds possible objects pairs, which may be semantically similar with respect to some proprieties, such as synonyms (name of attribute and class) of equal primary key attribute and equivalent of classes. The conforming is a variety of analysts assisted techniques that are used to resolve conflicts and mismatched objects.

The merging and restructuring phase generates an integrated schema from two component schemas that have been compared. The intermediate results are analyzed and restructured in order to eliminate the symmetrical and transitive relationship between objects.

In addition, we consider only one pair of schemas at a time; further, the result of integration schema is accumulated into a single schema, which evolves gradually towards the global object schema. For more details see (Malki, 2000).

5.4. *Validation*

The key question to be answered by validation is: how faithfully will a legacy database be represented by the global object schema produced from reverse engineering process? A schema is correct if all concepts of the underlying model are used correctly with respect to syntax and semantics (Chiang, 1994; Farhrner, 1995b; Hainaut, 1995). If all database instances that can be represented in the source schema can also be represented in the target database schema and vice versa, the process is information preserving.

Our validation process is completely interactive. Human intervention will be required in the verification and conformation of the results. In this process, that is not determinist, some conceptual construction will be examined either to be validated or to be put back in reason. However, it consists in identifying the hidden objects, reiterating the restructuring process in order to eliminate the symmetry and the transitivity of composition and inheritance relationships and insuring the consistency of the global object-oriented schema. For more details, see (Malki, 2000).

6. Conclusion

A methodology for performing reverse engineering of a relational database towards an object-oriented system is proposed. Our approach that requires the interaction with users (i.e., analyst or designer) uses the information extracted from form structures and instances as machine-analyzable source for database reverse engineering.

This methodology does necessarily a model that allows abstracting database forms and a framework that allows to analyzing forms in order to extract structure information as well as domain semantics. Forms are used in combination with other semantic sources, especially relational database schema and uses “head-knowledge”. The issues with regard to the validation and verification of the extraction and transforming process are also discussed.

An experimentation of this method has been achieved in the setting of the “Personnel Management” system of our university. This approach can supplement existing database reverse engineering techniques where forms constitute important uses of the database.

There are a number of possible extensions to this research:

– we have used techniques of datamining for identifying inheritance relationships; these techniques can be generalized for extracting data dependencies;

– for the migration toward Object-oriented systems, we have made only the extraction of the static aspect (i.e., data semantic and conceptual structure), it remains to extract the dynamic aspect in order to translate data and programs into Object-Oriented system.

Acknowledgments

We would like to thank the anonymous referees for their insightful comments and suggestions for improving this paper. This research was supported in part by National Scientific Research Council, Algeria, under contract No: B2205/-/11/99.

References

Anderson, M. (1994). Extracting a E.R. schema from a relational database through reverse engineering. In *Proc. of the 13th Inter. Conf. on the ERA'94*, pp. 403–419.

Armstrong, W. (1974). Dependency structures of data base relationships. *IFIP Congress*, 580–583.

Ayache, M., A. Flory (1995). A generation process of object-oriented database from E/R schema. In *Proc. of Int. Conf. of Soft. Eng. & Know. Eng.*, Rockville Maryland.

Batini, C. *et al.* (1984). A methodology for conceptual schema design of office database. *Information System*, **9**(3).

Batini, C., M. Lenzerini, S.B. Navathe (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, **18**.

Batini, C., S. Ceri, S.B. Navathe (1992). *Conceptual Database Design: an Entity Relationship Approach*. Benjamin Cummings

Behm, A. *et al.* (2000). Algebraic database migration to object technology. In *Proc. of Int. Conf. ER'00*, pp. 440–453.

Casanova, M.A., J.E.A. De Sà (1983). Designing entity-relationship schemas for conventional information system. In *Proc. of 3th Conf. On the Entity-Relationship Approach California*, pp. 265–277.

Castellanos, M., F. Saltor (1993). Extraction of data dependencies. In *Proc. of 3th European-Japanese Seminar on Information Modelling and Knowledge Bases*.

Chiang, R.H.L., T.M. Barron, V.C. Story (1994). Reverse engineering of relational databases: extraction of an EER model from a relational database. *Data and Knowledge Engineering*, **10**(12).

Choobineh, J. (1992). A form-based approach for database analysis and design. *Communication of the ACM*, **35**(2).

Elmasri, R., S. Navathe (1994). *Fundamentals of Database Systems*. Benjamin Cumming, 2nd edition.

Farhrner, C., G. Vossen (1995). A survey of database design transformation based on the entity-relationship model. *Data and Knowledge Engineering*, **15**, 213–250.

Farhrner, C., G. Vossen (1995). Transforming relational database schemas into object-oriented shemas according to ODMG-93. In *Proc. of the 4th Inter. Conf. on Deductive and Object-Oriented Database*. Singapour. pp. 429–446.

Hainaut, J.L., V. Englebert, J. Henrard, J.M. Hict, D. Roland (1995). Requirements for information system reverse engineering support. In *Proc. of the IEEE Working Conf. on Reverse Engineering*. Toronto.

Ji, W. (1991). An algorithm converting relational schemas to nested entity-relationship schemes. In *Proc of 10th int. Conf. on ERA'91*. San Mateo. pp. 231–246.

Johanneson, P. (1994). A method for translating relational shemas in to conceptual schemas. In *Proc. of the 10th IEEE int. Conf. on Data Engineering*. Houston.

Lefkovitz, H.C. (1979). A status report on the activityies of the CODASYL end user facilities committee (EUFC). *SIGMOD Rec.*, **10**(2).

Kalman, K. (1991). Implementation and critiques of an algorithm which maps a relational database to a conceptual model. In *Proc. of the 3th Inter. Conf. on Computer Aided Sotware Engineering*.

Mfourga, N. (1997). Extracting entity-relationship schemas from relational databases: a form-driven approach. In *Proc. of Working Conf. on Reverse Engineering WCRE'97*.

Malki, M., M. Ayache, M.K. Rahmouni (1999). Rétro-ingénierie des Bases de Données Relationnelles: Approche Basée sur l'Analyse de Formulaires. In *Proc.of Colloque of INFORSID'99*. Toulon, France.

Malki, M., A. Flory, M.K. Rahmouni (2000). Rétro-ingénierie des applications de gestion à partir de l'analyse des formulaires. *Research Report LRISI*, 04/2000 Univ. Sidi Bel Abbes.

Mannila, H. *et al.* (1994). *The Design of Relational Databases*. Addison-Wesley publishing.

Markowitz, V.M. (1990). Identifying extented entity-relationship object structures in relational schemas. *IEEE Trans. on Soft. Engineering*, **16**(8).

Navathe, S.B., A.M. Awong (1987). Abstracting relational and hierarchical data with a semantic data model. In *Proc. of 6th Int. Conf on ERA'87*, pp. 303–336.

Petit, J.M., F. Toumani, J. Kouloumdjian (1995). Relational database reverse engineering: a method based on Query analysis. *Inter. Journal of Cooperative Information System*, **4**(2,3), 287–316.

Piatetsky-Shapiro, G., W. Frawley (1991). *Knowledge Discovery in Databases*. AAAI Press/ The MIT Press, California.

Premerlani, W.J., M. Blaha (1994). An approach for reverse engineering of relational databases. *Communication of ACM*, **37**(5), 42–49.

Ramannathan, S., J. Hodges (1997). Extraction of object-oriented structures from existing relational databases. *SIG MOD Record*, **26**(1).

Shu, N.C. (1985). FORMAL: A forms-oriented, visual-directed application development system. *Computer*, **18**(8).

Soutou, C. (1998). Relational database reverse engineering: algorithm to extract cardinality constraints. *Data & Knowledge Engineering*, **28**, 161–207.

Tari, Z. *et al.* (1997). The reengineering of relational databases based on key and data correlation. In *Proc. of Int. Conf. of IFIP'97*. Published by Chapman & hall.

Ullman, J.D. (1984). *Principles of Databases Systems*. Computer sciences press.

Yao, S.B. *et al.* (1984). Formanager: an office forms management system. *ACM trans. On. Information System*, **2**(3).

M. Malki received the engineer degree in computer science from National Institute of Computer Science Algiers, and the Master of Science Degree in Computer Science from University of Sidi Bel-Abbes Algeria, in 1983 and 1992 respectively. He is currently a Ph.D. candidate in the Computer Science Department at the Sidi Bel-Abbes University. He is a senior lecturer in the Computer Science Department at the Sidi Bel-Abbes University. His research interests include semantic data modeling, database reverse engineering, information system reengineering, and building web-based applications.

A. Flory received the Ph.D. degree in computer science from Claude Bernard University, Lyon France, in 1977. He is a professor in the Computer Science Department at the National Institute of the Applied Sciences of Lyon . His research interests are engineering of information system, database reverse engineering, documentary information engineering, medical information systems.

M.K. Rahmouni received the Ph.D. degree in operational research from Southampton University UK, in 1987. He is a professor in the Computer Science Department at the University of Oran Algeria. His research interests include formal specifications, methodologies of specification in the information systems, database reverse engineering.

Objektinių schemų formavimas iš esamų realiacinių bazių vadovaujantis formomis

Mimoun MALKI, André FLORY, Mustapha Kamel RAHMOUNI

Straipsnyje nagrinėjamas uždavinys, kaip, vadovaujantis vartotojui pateikiamomis rezultatų formomis, atkurti reliacinių duomenų bazių schemas. Sprendžiant šį uždavinį siūloma pasinaudoti ir rezultatų formų struktūra, ir jų turiniu (konkrečiais duomenimis). Šitaip galima atkurti realiacinių bazių poschemius ir jų ribojimus. Gautus realiacinius poschemius siūloma transformuoti į objektinius poschemius ir, agreguojant objektinius poschemius, suformuoti atitinkamos duomenų bazės visą objektinę schemą.