Reverse Engineering User Interfaces for Database Conceptual Analysis

Ravi Ramdoyal, Anthony Cleve,
Anne-France Brogneaux, Jean-Luc Hainaut

Laboratoire d'Ingénierie des Bases de Données
Faculté d'Informatique, Université de Namur
21 rue Grandgagnage - 5000 Namur, Belgique
{rra,acl,afb,jlh}@info.fundp.ac.be

Résumé :

La première étape des méthodologies de conception de base de données consiste à éliciter les besoins à partir, entre-autres, dŠinterviews des utilisateurs. Ces besoins sont formalisés au sein dŠun schéma conceptuel du domaine dŠapplication, typiquement un diagramme Entité-Relation (ER). Le processus de validation de ces besoins demeure pourtant problématique car la représentation visuelle du modèle ER ne permet pas toujours une bonne compréhension par les utilisateurs finaux. En revanche, des prototypes dŠinterfaces utilisateurs peuvent être utilisés comme un moyen plus efficace pour exprimer, capturer et valider des besoins en termes de données. Considérant ces interfaces comme une vue physique de la base de données à concevoir, il est possible d'en dériver le schéma conceptuel sous-jacent en combinant des techniques de rétro-ingénierie, de transformation et d'intégration de schémas. Cet article présente les fondements d'une approche interactive et outillée mettant en oeuvre ces principes.

Mots-clefs :

Conception de systèmes d'information, Conception de bases de données, Analyse conceptuelle, Rétro-ingénierie d'interfaces homme-machine.

1 Introduction

In the realm of Requirements Engineering, data modeling plays a pivotal role, as it defines the semantic core of the future application. Accurately eliciting and validating user requirements is therefore vital to build a reliable documentation of the data application domain, and therefore a reliable information system.

Database engineering precisely focuses on data modeling, where these requirements are typically expressed by means of a conceptual schema (or model), that is, an abstract view of the application domain. Designing databases for data-intensive business applications most often relies on requirements elicitation techniques such as the analysis of corporate documents and interviews of stakeholders. Various techniques therefore exist to elicitate data requirements, but they usually do not actively and interactively involve end-users.

Still, the necessity to actively involve end-users of a future IT system during its specification and development steps has long been advocated [1, 2]. Involving end-users in the expression of their needs and in the definition of an appropriate solution may allow to avoid resistance toward a new information system infrastructure, as well as to stimulate productivity [3]. Besides, as a matter of fact, end-users know "how business is done" in the environment for which software is being developed. They know the qualities and the flaws of the information systems currently used, and therefore have the ability to state what could be done to improve it [4, 5].

As for the validation of these data requirements, their formal graphical representation is often difficult to apprehend for the end-users. Indeed, while the analysts focus on building requirements meeting various expectations (correctness, completeness, consistency, ...), requirements from the end-users standpoint need to be understandable and expressive enough.

In order to tackle this issue, we present a tool-supported approach to elicitate and validate database requirements, based on end-users involvement through interactive prototyping, and using well-mastered techniques coming from various fields of study. By taking advantage of their expressiveness and understandability, we propose the use of form-based user interfaces as a two-way channel to efficiently express, capture and validate data requirements with end-users.

In particular, we capitalize on the transformational power of reverse engineering techniques, which aim at extracting specifications from existing ar-

tifacts such as the DDL code of the database, the source code of application programs, data instances and so on.

The remaining of the paper is structured as follows. Section 2 describes our research context based on a related work discussion. The main principles of our proposal are detailed in Section 3. In Section 4 we elaborate on the key specificities of our approach, namely reverse engineering, view integration and transformational approach. Section 5 briefly presents the RAINBOW tool kit supporting the proposed methodology. Finally, in Section 6, we discuss the merits and limitations of our proposal and anticipate future work.

2 Research Context

2.1 Data Requirements Elicitation and Validation

The process of designing and implementing a database that has to meet specific user requirements has been described extensively in the literature [6] and has been available for several decades in CASE tools. It consists of four main subprocesses :

(a) *Conceptual design* which aims at expressing user requirements into a conceptual schema, that is, a technology-independent abstract specification of the future database. Such a schema is also known as a *PIM* (Platform-Independent Model) in *MDE* (Model Driven Engineering).

(b) *Logical design*, which produces an operational logical schema (Platform-Specific Model or *PSM* in MDE) that translates the constructs of the conceptual schema according to a specific technology family without loss of semantics (e.g. Relational, XML, Object-Oriented).

(c) *Physical design*, which augments the logical schema with performance-oriented constructs and parameters, such as indexes, buffer management strategies or lock management policies.

(d) *Coding*, which translates the physical schema (and some other artifacts) into the DDL (*Data Definition Language*) and procedural fragments of the database management system.

In this paper, we naturally focus on the first step of the database engineering process, i.e., conceptual design. More particularly, we propose techniques that allow to reduce the potential gap between actual user requirements and their translation in the conceptual schema.

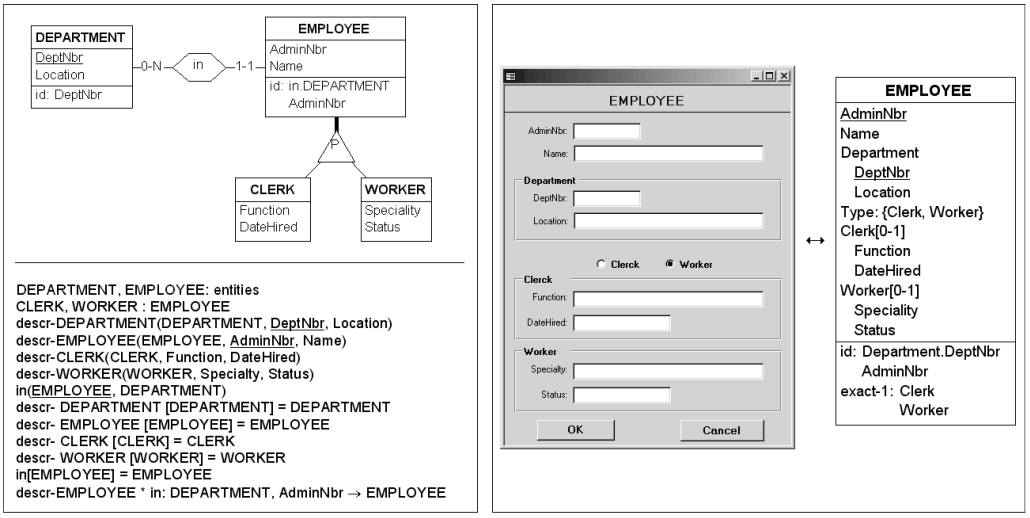

FIG. 1 – (a) An ER schema and its formal expression. (b) An electronic form and its informational contents.

Among database models, the Entity-Relationship (ER) model has long been considered to be the best medium to express conceptual requirements. Its simplicity, its graphical representation, the availability of numerous CASE tools that include an ER schema editor (should) make it the ideal communication medium between designers and users.

However, this statement has proved over-optimistic in many situations. It appears that the ER formalism, despite its merits, often fails to meet its objectives as an intuitive and reliable communication medium in which end-users are involved. The reason is easy to grasp : a conceptual schema is just a graphical presentation of a large and complex set of 1st and 2nd order predicates. Fig. 1(a) shows a small conceptual schema and its NF^2 relational interpretation according to the GER formalism [7]. The intrinsic complexity of the requirements has been concealed by the apparent intuitiveness of the ER graphical notation but has not disappeared. An in-depth comprehension of an ER schema implies the understanding of such non trivial concepts as sets, non-1st normal form relations, algebraic operators, candidate keys and functional dependencies.

Paradoxically, most users are quite able to deal with complex data structures, provided they are organized according to familiar layouts. In particular, those users interact daily with computer systems through electronic forms,

which have proved to be more natural and intuitive than usual conceptual formalisms to express data requirements [8] while making the semantics of the underlying data understandable [9]. Indeed, such documents present data as a two (and a half)-dimensional arrangement of data elements that mimics the way users perceives information in the real world, and in particular in paper forms and documents. Hence the idea to use these forms as the preferred way to describe data structures in requirements engineering processes.

2.2 Reverse Engineering Prototype User-Interfaces

As we have seen, there is a proved link between graphical interfaces and data models, which is usually exploited in a forward engineering perspective. Indeed, implementing a database from its conceptual schema is a well-mastered process that has long been studied in the database research community and applied in industry. Once the requirements have been elicited and the database conceptual schema drawn, transformational techniques allow to automate the production of logical and physical schemas from their conceptual counterpart [7], and even to produce the artifacts of the final application : interfaces, programs, database code, etc.

Conversely, a form contains data structures that can be seen as a particular *view* of the conceptual schema, since the transition from forms to a semantic model has been shown to be tractable [10]. Therefore, if a form is a concrete implementation of a part of the conceptual schema, database reverse engineering techniques can be applied to recover that part of the schema.

These techniques can hence be combined with prototyping, which has long been recognized a valuable requirements elicitation (RE) and validation technique. It is typically used for elicitation when early feedback from stakeholders is needed [11]. Prototype user interfaces (PUI) may also act as a basis for other RE techniques such as interviews or group elicitation [12].

Deriving requirements from prototype artifacts is obviously not a novel idea in itself. Back in 1984, Batini and al. already studied paper forms as a widely used mean to collect and communicate data in the office environment. Later on, Choobineh *et al.* [8] explored a form-based approach for database analysis and design, and developed an analyst-oriented Form Definition System to create forms as well as an Expert Database Design System that incrementally produced an ER diagram based on the successive analysis of a set of forms.

We can also mention Schneider [13], who proposed a general strategy to

extract crucial pieces of knowledge from a prototype and from its developer, as well as Kösters *et al.* [14] who introduced a requirements analysis method combining user interface and domain requirements analysis. The first stage of the latter method develops an initial domain model and a task model describing the user activities accomplished with the help of the system, through standard elicitation techniques. During a second step, the task model served as a basis for the completion of the domain model as well as for the development of a user interface model. This combined approach is similar to ours, but the initial domain model is developed.

More recently, Ravid and Berry [15] suggested a method for fine-tuning the process of prototyping. They identified the categories of requirements information that a prototype user interface may contain. One of these categories concerns *"the application's data model, data dictionary, and data-processing capabilities"*, which is our main concern.

3 Proposal

As we have seen, providing a better requirements acquisition process for database engineering implies bridging the gap between end-users and analysts. Since the traditional ER schema has shown understandability limitations, this issue clearly calls for a better medium, which should be common to all the stakeholders and rich enough to convey relevant meaning and interactivity. For this purpose, we propose to use form-based user interfaces as a two-way channel to efficiently capture and validate data requirements with end-users, while providing the end-users with adequate tools to transparently draw by themselves the interfaces describing the underlying key concepts of their application domain. Bearing a little training and as previously explained, involving end-users in such processes may have a very positive impact. This is especially true in the RE process for which it is essential to avoid mismatches between the actual needs of end-users and the way they are formalized.

Since existing artifacts can be used to recover the underlying requirements through well-mastered reverse engineering techniques, we advocate to use such techniques in forward engineering by working with the virtual artifacts produced by the end-users. This approach benefits from the advantages of rapid prototyping (such as visual expressiveness, early feedback and clarification) [16, 15], while making the user a central actor of the process.

Besides, these principles are at the foundation of broader approaches, such as the ReQuest framework [17], which provides a complete methodology and a set of tools to deal with the analysis, development and maintenance of web-based data-intensive applications. The ReQuest framework deals with data modeling as well as the dynamic aspects of the future application (such as task analysis, behavior of the application, etc.), while providing generators for several components of the future application (database, framework skeleton, etc.).

In this paper, we focus on the specificities of our alternative RAINBOW approach, which keeps the same overall philosophy but focuses on the specification of data requirements as part of a greater requirements engineering process. It intends to involve the end-users in a simple and interactive way while providing the analysts with semi-automatic tools. The approach is therefore formalized into a seven-step process whose aim is not to provide a ready-to-use application, but a set of specification documents and tools, in order to support the development of future applications.

Let us illustrate these steps with a simple example : the context is the development of a tailored IT solution to manage a small sales company. In this company, the information on clients are traditionally stored, as well as all the orders that they submitted. Each order is created in a given store and specifies a list of products.

Represent : First of all, the *end-users* are invited to *draw form-based interfaces* describing each key concept of their application domain. Such interfaces are typically an encoding screen, such as a window to introduce a new registered customer into an hypothetical system. During this phase, the end-users must provide details on the properties of the concepts (size of a field, expected type of values, possible default or predefined values, ...) and the links between them. Let us keep in mind that the objective here is *not* to lead the end-users to draw the interfaces of a future application, but to express requirements through a medium that is familiar to them. This is done using a dedicated drawing tool kit which provides them with a limited set of simple but usual form widgets (Fig. 2) such as *interfaces* ; *tables* and *group boxes* ; *text input* fields and *selection fields* (list, radio buttons, check boxes, ...) ; *button panels* (to define links between the interfaces). Fig. 3(a) illustrates four key concepts (CUSTOMER, ORDER, PRODUCT, STORE) that the end-users might draw for our example.

Adapt : Once the interfaces are drawn, we apply *database reverse engineering techniques* to recover the underlying conceptual schema of the domain.

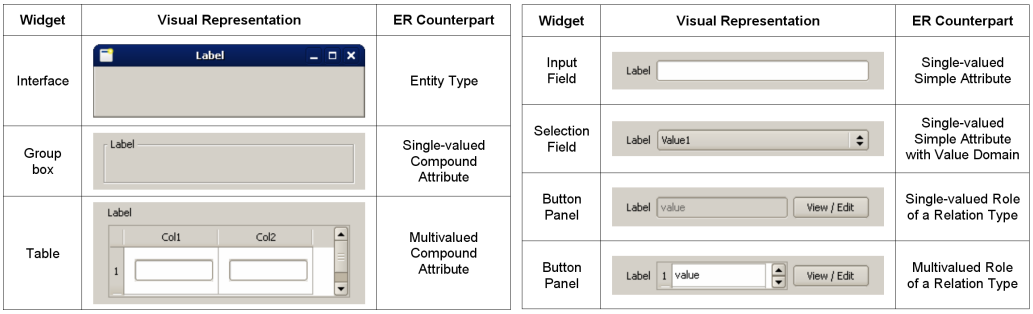

FIG. 2 – Available graphical widgets with their mapping rules to data structures.

More specifically, the interfaces are firstly automatically analyzed to extract data models using the mapping rules of Fig. 2. Secondly, each individual entity type (Fig. 3(b)) is transformed into an independent logical schema by recursively transforming each compound attributes (Fig. 3(c)). As we can see, the structure of the data models is very simple, but given the nature of the transformations that we use, there is no loss of semantics.

Investigate : Afterward, cross-analyzing each individual schema usually brings to light possible ambiguities as well as redundant information contained in the interfaces. In particular, semantic and syntaxic redundancies are automatically identified and presented to the end-users for manual validation, in this step crucial for a future integration of the individual schemas.

Nurture : Once the interfaces are validated by both the *analysts* and *end-users*, the next step is to elicitate possible constraints among them. For this purpose, we analyze the data samples provided by the end-users to highlight functional dependencies and enrich our semantic knowledge using instances, as long advocated by Tseng and Mannino [18] or Ram [19]. The first step hence consists in gathering positive and negative *data samples* from the end-users by using the the very same interfaces they drew, in order to populate their underlying data structure. The second step consists in mining functional dependencies from the data samples. Several algorithms have been proposed to support such a task [20]. We are currently using a simplified version of such a mining algorithm to identify candidate dependency constraints, which are submitted to the end-users for validation.

Bind : The validated redundancies and constraints are finally processed to integrate the individual schemas into a pure conceptual schema that represents the data requirements. Transformational techniques have proved to be

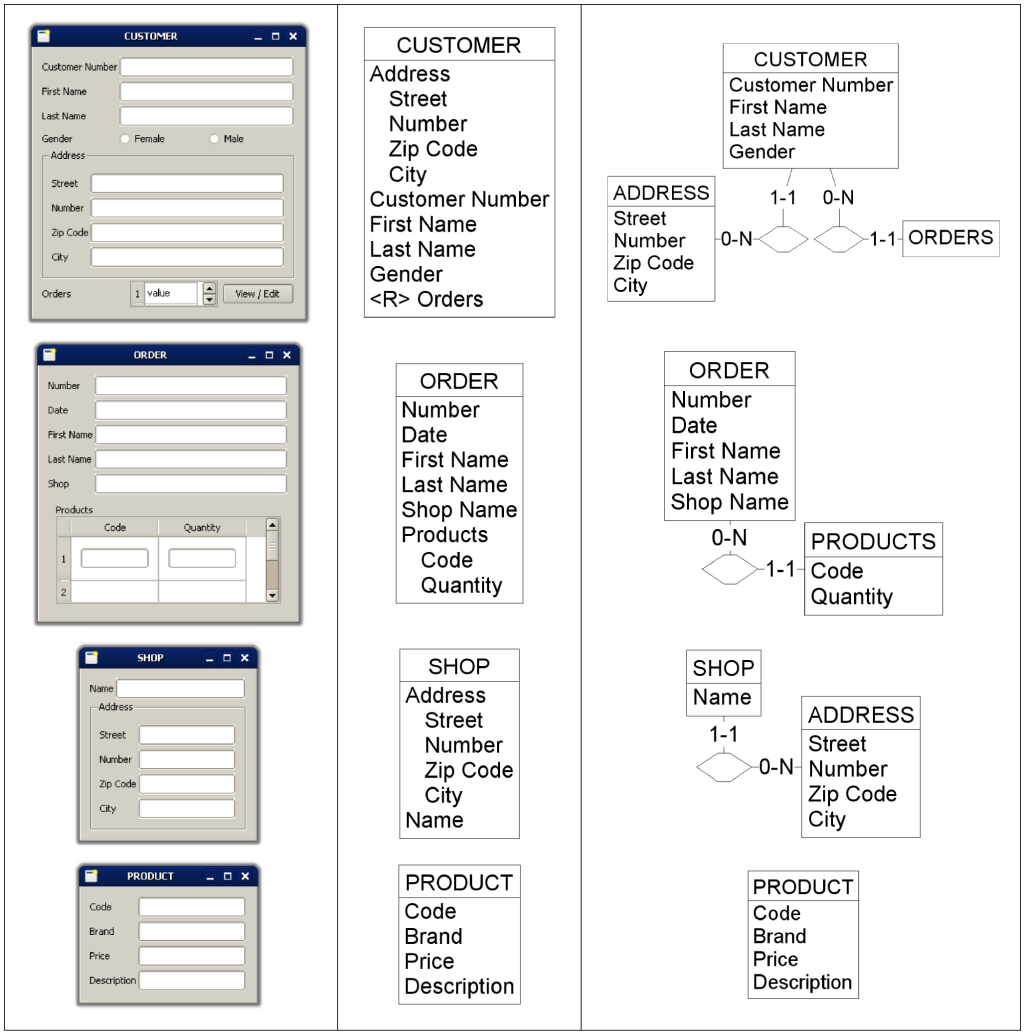

FIG. 3 – (a) Example of user-drawn interfaces. (b) Translation of the interfaces into raw entity types. (c) Translation of the raw entity types into independent schemas.

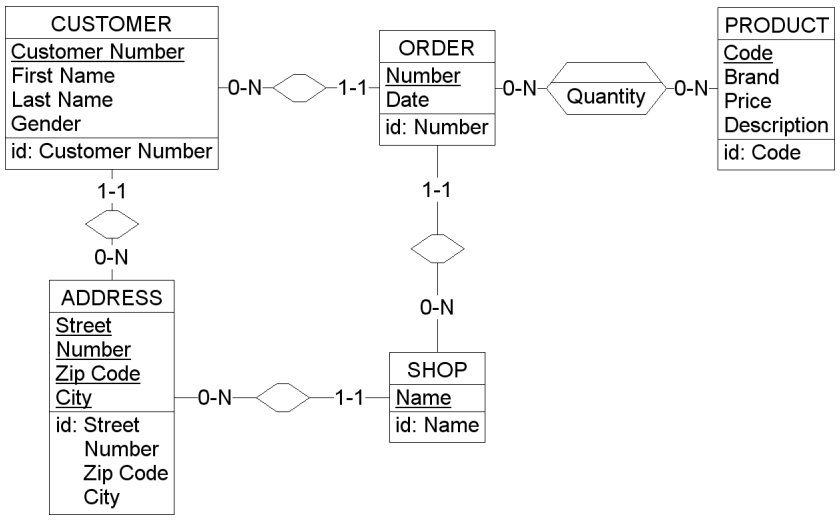

FIG. 4 – Integrated schema of our running example.

particularly powerful to carry out this process, which is a typical case of *database schema integration* [21]. They enable to integrate similar objects into a unique, non-redundant structure, without any loss of semantics. Fig. 3(b) illustrates the result of the integration for our running example.

Objectify : When all the constraints are settled, a lightweight prototype application with its underlying database is generated. This prototype is a simple data manager that uses the interfaces drawn by the end-users and allows to navigate through the concepts that have been expressed, typically to query, create, modify and remove data.

Wander : Finally, the end users are invited to *"play" with the prototype* in order to ultimately validate the requirements, or identify remaining flaws.

4 Methodological Specificities

Discussing the validity of our approach is beyond the scope of this paper and has been discussed elsewhere. We will rather focus on presenting the main specificities of the RAINBOW approach.

4.1 Reverse Engineering

Reverse engineering is a well known fundamental aspect of Database Engineering, which consists, among other things, in recovering or reconstructing the functional specifications from a piece of software, starting mainly from the source code of the programs [22, 23], typically when an existing database has to be restructured or migrated toward a different technology. Reverse engineering hence aims at recovering a conceptual schema that is the most faithful to the original one, working from multiple system artifacts, such as : documentation (when available), the DDL code of the database, data instances, screens, reports and forms, source code of application programs.

However, our objective is here to "build the truth" rather than "find the truth", as in traditional reverse engineering situations. In particular, the interfaces are used as a language of specification as opposed to the usual reverse engineering of existing screens.

4.2 View Integration

The first type of problems is the *semantic redundancy and ambiguity*. This issue arises due to the limitations of written natural language (paronymy, similarity, polysemy/homography, synonymy), which lead to unclear labels in the interfaces. When adding possible spelling mistakes, the necessity to clarify the labels of the interface becomes indisputable. In our example, one can for instance notice the closeness of the labels "Orders" (from the interface "CUSTOMER") and "ORDER".

Identifying similar labels is a well-known problem that can be dealt with using *String Metrics* (a.k.a. *String Distances* [24]). By using such reliable metrics, such as Jaro-Winkler's distance [25], one can identify ambiguous elements within user-drawn interfaces. Detailing the technical specificities of each stage of our approach is beyond the scope of this paper, however let us describe this current matter to illustrate the *modus operandi* that we use for each step requiring user validation.

First of all, for a given string distance metric sdm and threshold t, we define two labels l_1 and l_2 as *lexically similar* iff $sdm(l_1, l_2) \leq t$. Within a set of labels $L = \{l_1, l_2, ..., l_n\}$, we also define a subset L_l of lexically similar labels as a subset $\{l_{l_1}, l_{l_2}, ..., l_{l_m}\} \subset L$ verifying (a) $\forall l_{l_i}, l_{l_j} \in L_l : sdm(l_{l_i}, l_{l_j}) \leq t$, and (b) $\forall l_{l_i} \in L_l, l_k \in L \neg L_l : sdm(l_{l_k}, l_k) > t$.

Since we want to partition the set L of all the labels available in the user-

drawn interfaces into subsets of lexically similar labels, we use Algorithm 1 to build the set of the lexically similar subsets, then visually point out the discovered similarities between concepts in the user-drawn interfaces in order ask the end-users to to validate or reject them.

Algorithm 1 Build the set of lexically similar subsets for a given set of labels

Require: $L = \{l_1, l_2, ..., l_n\}$
  $L_l \Leftarrow \emptyset$
  for $i = 1$ to n **do**
    if $\exists L_{l_k} \in L_l : l_i \in L_{l_k}$ **then**
      $L_{l_i} \Leftarrow L_{l_k}$
    else
      $L_{l_i} \Leftarrow \{l_i\}$
      $L_l \Leftarrow L_l + L_{l_i}$
    end if
    for $j = i + 1$ to n **do**
      if $sdm(l_i, l_j) \leq t$ **then**
        $L_{l_i} \Leftarrow L_{l_i} \cup \{l_j\}$
      end if
    end for
  end for

Another issue concerns *structural redundancy*. A structural redundancy occurs when two sets of attributes are said to be similar, such as "First Name" and "Last Name" which are shared by the interfaces "CUSTOMER" and "ORDER". The similarity between pairs of attributes from each set is measured using a set of indicators (typically the label). For each indicator, we define a *similarity index*, the values of which fall between 0 (strictly different) and 1 (strictly identical). The similarity of two attributes is therefore a weighted average of the similarity indicators taken into account.

Given the hierarchical structure of the interfaces, and thus the tree-like structure of the underlying models, the problem of extracting such structural redundancies is actually a particular case of *frequent embedded subtrees mining in rooted unordered trees*. This well-known complex issue is described for instance by Jimenez et al. [26], who also lists subsequent algorithms such as Zaki's SLEUTH or Asai et al.'s UNOT. As for the semantic similarities, we first search for the structural redundancies, using such an algorithm, then visually point them out. This is done in order to ask the end-users to arbi-

trate them, which means classifying the relation between the concepts lying behind two structurally similar interfaces among one of these most usual cases : equality, union, comprehension, complementarity or difference. Further and related details on the technical specificities of this matter can be found in [17].

For simplicity, we consider in this paper that two similar objects refer to the same concept and can therefore be merged. Hence, the entity types of the logical schemas are integrated by pairs, and whenever needed, end-users are invited to choose which attributes of the two objects are relevant and should thus be kept during the merging process. However, there is not always a strict identity between two concepts and other integration techniques must then be used to resolve redundancies [27].

4.3 Transformational Approach

Fig. 1(b)(left) shows an electronic form of a complex data structure derived from the conceptual schema of Fig. 1(a). Its informational contents is represented at the right-hand side as a record type, each field of which represents either an elementary form field or a possibly multivalued grouping of such fields. From the theoretical point of view, there is no formal guarantee that both schemas are equivalent, that is, that the record type of Fig. 1(b) provides the same information as the schema of Fig. 1(a). Although they are not *exactly* equivalent, we can show that the record type captures most of the information of the conceptual schema. For this, we will use a demonstration based on a transformational approach. According to the latter, every engineering process can be modeled as a chain of schema transformations [7]. A transformation operator is defined by a rewriting rule that substitutes a target schema construct for a source construct. The most interesting operators are semantics-preserving, in that the source and target constructs convey the same semantics (they have the same meaning though presented differently).

Fig. 5 illustrates two important semantics-preserving operators, namely *attribute to entity type mutation* and *upward inheritance*. The first one (T1) transforms an entity type into an equivalent attribute (and conversely). The second transformation (T2) integrates the subtypes of an entity type as complex attributes of the latter (and conversely). They lack some necessary pre- and post-conditions to be fully semantics-preserving, but there are sufficient considering the scope of this paper.

Now, we can apply transformation T1 on entity type DEPARTMENT

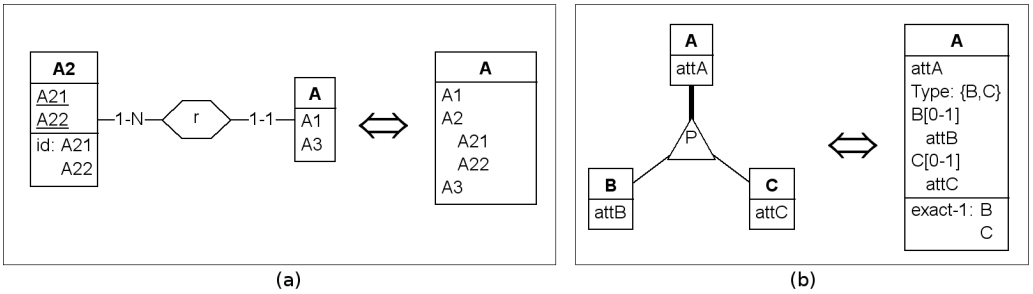

FIG. 5 – T1 (a) and T2 (b), two (almost) semantics-preserving transformations

of Fig. 1(a), which yields the compound attribute Department in Fig. 1(b). Then, we apply transformation T2 on the subtypes CLERK and WORKER of the schema of Fig. 1(a). They are transformed into attributes Clerk, Worker and Type of the schema of Fig. 1(b). Since the transformations are semantics-preserving, they can be applied in the reverse way, in such a way that the schema of Fig. 1(b) can be transformed in that of Fig. 1(a).

As a conclusion, we can consider that the electronic form of Fig. 1(b) expresses in an intuitive way the information requirements formally expressed in Fig. 1(a).

5 Tool Support

The RAINBOW Tool Kitis a user-oriented development environment, intended to assist the end-users in the definition and validation of database requirements through prototyping. It therefore supports the first steps of our approach by offering ready-to-use widgets and mapping rules with the ER model.

The tool kit interacts with the repository of DB-Main, which is a database engineering CASE Tool [28] providing all the necessary functionalities to support a complete database design process (from the definition of the conceptual schema to the generation of the database code). It notably provides transformation tools and supports database reverse engineering. The interaction between these tools allows the cover the whole database engineering process either from the end-users's perspective or from the analyst's point of view.

6 Concluding Remarks

6.1 Contributions

In this paper, we have presented an interactive visual approach to bridge the gap between end-users and analysts during the requirements analysis phase of the database engineering process. This approach relies on the expressiveness and understandability of form-based user interfaces, used jointly with well-mastered reverse engineering techniques to acquire data specifications from existing resources, in order to efficiently capture and validate data requirements with end-users.

We offer a very simple interface model, inspired by high level abstract models such as UsiXML [29], TERESA [30] or the Teallach classification [31]. Since any other interface widget dedicated to data representation can be expressed using these basic components, the decision of limiting the number of available widgets appears to simplify the user interaction rather than limiting it. Indeed, the limited number of widgets makes the drawing process easier for end-users, since they are not overloaded with information.

The RAINBOW approach lies within broader perspectives, such as the ReQuest Framework. While offering a precious user empowerment and involvement in the data requirements elicitation, we provide an expressive and interactive part of user requirements thanks to the RAINBOW Tool Kit and its connection to the DB-Main CASE Tool. The requirements are materialized as a documented application domain (the conceptual model) and a documented database (DDL, queries, etc.). The simple yet operational generated prototype can serve as a basis for further application development over the database.

6.2 Limitations

Although our approach addresses a significant subset of data requirements, it does not replace more traditional task and information analysis approaches. It rather complements them. For instance, in order to efficiently guide the end-users during their participation, the analysts may want to go through the traditional steps of interviews, document analysis and use cases definitions in order to elicitate an early list of key concepts that he could submit to the end-users.

Regarding the ability of end-users to create form-based interfaces them-

selves, it must be pointed out that the understanding of user interfaces has significantly improved in organizations thanks to the increasing use of and training in IT. From this observation, we conclude that users are now quite familiar with graphical user interfaces (GUIs), and bearing a short training period to gain minimal knowledge in interface design, many of them should be able to draw a low-fidelity prototype of the interfaces for the future application [4].

Another major concern lies within the *Nurture* phase, which may need large data samples (i.e., many instances) in order to discover any useful patterns, while the users are likely to provide a limited number of data instances.

6.3 Future work

As discussed above, it is obvious that with proper training the targeted end-users (i.e., accustomed to computer manipulation and form-based interfaces) should be able to represent simple and well-known concepts. However, while limiting the possible data structures to simple ones can be seen as a "positive" simplification, we intend to push our investigation further to allow the users to also express complex data structures such as temporal or semi-structured data.

In addition, a more general issue concerns the feasibility of involving neophytes in our approach. Although a software engineering process implies that the future users will interact with a computer application, the stakeholders involved in the requirements phase may be only casual users, or even novices. To ensure the quality of the requirements, these less proficient users can certainly not be cast away and must be provided with adequate means to express their needs. We therefore intend to work on our approach to make it even more intuitive, and therefore accessible to neophytes.

Références

[1] Schuler, D., Namioka, A., eds. : Participatory Design : Principles and Practices. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA (1993)

[2] Beyer, H., Holtzblatt, K. : Contextual design : defining customer-centered systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)

[3] Vosburgh, J., Curtis, B., Wolverton, R., Albert, B., Malec, H., Hoben, S., Liu, Y. : Productivity factors and programming environments. In : ICSE. (1984) 143–152

[4] Fischer, G. : Beyond 'couch potatoes' : From consumers to designers and active contributors. First Monday **7** (2002)

[5] Illich, I. : Tools for Conviviality. Harper & Row Publishers, New York (1973)

[6] Batini, C., Ceri, S., Navathe, S.B. : Conceptual database design : an Entity-relationship approach. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA (1992)

[7] Hainaut, J.L. : The transformational approach to database engineering. In : Generative and Transformational Techniques in Software Engineering. Volume 4143 of LNCS. (2006) 95–143

[8] Choobineh, J., Mannino, M.V., Tseng, V.P. : A form-based approach for database analysis and design. Com. of the ACM **Vol. 35, Nř2** (1992) 108–120

[9] Terwilliger, J.F., Delcambre, L.M.L., Logan, J. : Querying through a user interface. Data Knowl. Eng. **63**(3) (2007) 774–794

[10] Rollinson, S.R., Roberts, S.A. : Formalizing the informational content of database user interfaces. In : Proc. of the 17th Int. Conf. on Conceptual Modeling (ER'98), Springer-Verlag (1998) 65–77

[11] Davis, A.M. : Operational prototyping : A new development approach. IEEE Softw. **9**(5) (1992) 70–78

[12] Nuseibeh, B., Easterbrook, S. : Requirements engineering : a roadmap. In : Proc. of the Conf. on The Future of Software Engineering, ACM Press (2000) 35–46

[13] Schneider, K. : Prototypes as assets, not toys : why and how to extract knowledge from prototypes. In : Proc. of the 18th Int. Conf. on Software Engineering (ICSE'96), IEEE Comp. Soc. (1996) 522–531

[14] Kösters, G., Six, H.W., Voss, J. : Combined analysis of user interface and domain requirements. In : Proc. of the 2nd Int. Conf. on Requirements Engineering, Washington, DC, USA, IEEE Computer Society (1996) 199

[15] Ravid, A., Berry, D.M. : A method for extracting and stating software requirements that a user interface prototype contains. Requir. Eng. **5**(4) (2000) 225–241

[16] Gomaa, H., Scott, D.B. : Prototyping as a tool in the specification of user requirements. In : Proc. of the 5th Int. Conf. on Software Engineering (ICSE'81), IEEE Press (1981) 333–342

[17] Vilz, J., Brogneaux, A.F., Ramdoyal, R., Englebert, V., Hainaut, J.L. : Data conceptualisation for web-based data-centred application design. In : Proc. of the Advanced Information Systems Engineering, 18th International Conference (CAiSE'06). LNCS (2006) 205–219

[18] Tseng, V.P., Mannino, M.V. : Inferring database requirements from examples in forms. In : Proc. of the 7th Int. Conf. on ER Approach. (1988) 391–405

[19] Ram, S. : Deriving functional dependencies from the entity-relationship model. Commun. ACM **38**(9) (1995) 95–107

[20] Yao, H., Hamilton, H.J. : Mining functional dependencies from data. Data Min. Knowl. Discov. **16**(2) (2008) 197–219

[21] Batini, C., Lenzerini, M., Navathe, S.B. : A comparative analysis of methodologies for database schema integration. ACM Computing Surveys **18**(4) (1986) 323–364

[22] Chikofsky, E.J., II, J.H.C. : Reverse engineering and design recovery : A taxonomy. IEEE Software **7**(1) (1990) 13–17

[23] Hall, P.A.V. : Software Reuse and Reverse Engineering in Practice. Chapman & Hall, Ltd., London, UK, UK (1992)

[24] Cohen, W.W., Ravikumar, P., Fienberg, S.E. : A comparison of string distance metrics for name-matching tasks. In : Proc. of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03). (2003) 73–78

[25] Winkler, W.E. : String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In : Proc. of the Section on Survey Research Methods, American Statistical Association. (1990) 472–477

[26] Jiménez, A., Berzal, F., Cubero, J.C. : Mining induced and embedded subtrees in ordered, unordered, and partially-ordered trees. In : Proc. of Foundations of Intelligent Systems, 17th International Symposium, ISMIS 2008, Toronto, Canada. (2008) 111–120

[27] Spaccapietra, S., Parent, C., Dupont, Y. : Model independent assertions for integration of heterogeneous schemas. The VLDB Journal **1**(1) (1992) 81–126

[28] DB-Main : The official website `http://www.db-main.be`.

[29] Limbourg, Q., Vanderdonckt, J. : Usixml : A user interface description language supporting multiple levels of independence. In : Proc. of ICWE 2004 Workshops. (2004) 325–338

[30] Mori, G., Paternò, F., Santoro, C. : Design and development of multidevice user interfaces through multiple logical descriptions. IEEE Trans. Softw. Eng. **30**(8) (2004) 507–520

[31] Griffiths, T., Barclay, P.J., McKirdy, J., Paton, N.W., Gray, P.D., Kennedy, J.B., Cooper, R., Goble, C.A., West, A., Smyth, M. : Teallach : A model-based user interface development environment for object databases. In : UIDIS. (1999) 86–96