Reverse Engineering User-Drawn Form-Based Interfaces for Interactive Database Conceptual Analysis

Ravi Ramdoyal

Laboratory of Database Application Engineering - PReCISE Research Center
Faculty of Computer Science, University of Namur, Belgium
rra@info.fundp.ac.be

Abstract. In this paper, we address the problem of eliciting, communicating and validating the static data requirements of a software engineering project, while improving the end-user involvement. For this purpose, given an environment for which electronic forms are a privileged way to exchange information and stakeholders familiar with form-based (computer) interaction, we propose to use form-based user-drawn interfaces as a two-way channel to interactively capture and validate static data requirements with end-users, by specializing and integrating standard techniques to help acquire data specifications from existing artifacts. Since the main principles of our approach are already presented in [1], we here focus on discussing two fundamental aspects of this research, namely the means to make end-users major stakeholders in the data requirements process, and the challenges facing the validation of such a transversal research.

Keywords: Information systems engineering, Requirements engineering, Database engineering, Human-computer interfaces reverse engineering.

1 Introduction

Requirements engineering is a key step in the realm of Software engineering, since its lays the ground work for further analysis, design and development. Within this process, Database engineering focuses on data modeling, where the static data requirements are typically expressed by means of a conceptual schema, which is an abstract view of the static objects of the application domain. Since long, conceptual schemas have proved to be difficult to validate by laymen, while traditional database requirements elicitation techniques, such as the analysis of corporate documents and interviews of stakeholders, usually do not actively and interactively involve end-users in the overall specification and development of the database. Still, the necessity to associate end-users of the future system with its specification and development steps has long been advocated [2]. In particular, the process of eliciting static data requirements should make end-users feel more involved and give them intuitive and expressive means to convey

their requirements to the analysts. Conversely, analysts should also be able to capture and validate these requirements by discussing them with the end-users.

In order to facilitate this communication, we present the tool-supported RAINBOW approach, which relies on reverse engineering user-drawn form-based interfaces to perform an interactive database conceptual analysis. This approach to elicit and validate database requirements is based on end-users involvement through interactive prototyping, and the adaptation of techniques coming from various fields of study. Since more details on our approach can be found in [1], the remainder of the paper is structured as follows. Section 2 briefly delineates the research context and related works. The main principles of our proposal are exposed in Section 3. In Section 4, we elaborate on the implications of an improved end-user involvement and the challenges facing the validation of our research. Finally, in Section 5, we discuss the merits and limitations of our proposal and anticipate future work.

2 Research Context

In our research, we focus on database conceptual modeling, through which user requirements are translated into a conceptual schema representing the application domain. The Entity-Relationship (ER) model has long been the most popular medium to express conceptual requirements [3], but this formalism often fails to act as an effective end-users communication medium because of its intrinsic complexity. Still, most users are quite able to deal with complex data structures that are expressed through more natural and intuitive layouts such as electronic forms [4].

This strong link existing between graphical interfaces and data models is usually exploited in forward engineering, by straightforwardly producing artifacts such as form-based interfaces from a conceptual schema, using transformational and generative techniques [5]. In particular, prototyping [6] often acts as a basis for interviews or group elicitation to provide early feedback. Conversely, a form contains data structures that can be seen as a particular *view* of a conceptual schema, which implies that Database reverse engineering [7] techniques can be applied to such interfaces to recover fragments of the conceptual schema.

Deriving requirements from prototype artifacts has a long tradition, but the number of studies on the subject is limited (especially recently) and several limitations must be underlined in most of them [1]. First of all, older studies do not intimately involve end-users in the database design process. More generally, the tools provided for the drawing of the interfaces are not dedicated to this purpose and/or not convenient for end-users. Secondly, the underlying form model of the interfaces must often be constructed by analyzing the layout of the form before its content. This is strongly related to the fact that the existing approaches also aim to create the final form-based interfaces of the future application. Regarding the coherence of these interfaces, it is assumed that the labels are used consistently through out the different forms, and little care is given to possible lexical variation (paronymy, feminine, plural, spelling, mistakes, etc.) and on-

tological ambiguity (polysemy, homography, synonymy). The use of examples (either through static statements or dynamic interaction) is not systematically used to elicit constraints and dependencies. And last but not least, these approaches do not use the form-based interfaces as a means for the analysts to validate the underlying integrated data model.

3 Proposal

Within the requirements engineering phase of a software engineering project, our research therefore addresses the combination of reverse engineering techniques with user-based prototyping in order to interactively involve end-users in the conceptual modeling of the application domain. More precisely, given an environment for which electronic forms are a privileged way to exchange information and stakeholders are familiar with form-based (computer) interaction, we propose to use form-based user-drawn interfaces as a two-way channel to interactively capture and validate static data requirements with end-users, in order to alleviate understandability limitations of the ER model.

To succeed, we need to overcome several challenges inherent to the involved fields and their combination. Regarding Database engineering, we notably need to clarify the terminology, elicit constraints and dependencies, handle schema integration and generate applicative components. Regarding Database reverse engineering, we must handle the extraction of data models from the form-based interfaces, and since we want to make the Prototyping interactive, we need to enable the expression of concepts through form-based interfaces and the testing of generated components. Finally, integrating these fields into an approach involving end-users implies managing this user implication and tailoring the techniques.

To handle these challenges and answer the concerns raised in Section 2, our RAINBOW approach is formalized into seven steps involving end-users in a simple and interactive fashion, using interfaces as a specification language rather than legacy artifacts (as in traditional Reverse engineering), and providing the analysts with semi-automatic tools. Let us give an overview of these steps, for which more details can be found in [1]: (1) *Represent*: After preliminary discussions and appropriate training, the end-users are invited to draw a set of form-based interfaces to perform usual tasks of their application domain. Such interfaces are typically entry forms to capture data on, say, a new customer or a new product. A dedicated drawing tool intentionally provides them with a constrained layout mechanism and a limited set of primitive widgets (namely *interfaces, group boxes, tables, input fields, selection fields* and *action buttons*). (2) *Adapt*: Once the interfaces are drawn, mapping rules are automatically applied to extract the underlying data models of the interfaces. (3) *Investigate*: The end-users and the analyst then jointly cross-analyze the interfaces to arbitrate the possible labeling ambiguities (lexically or ontologically similar labels) and structural similarities (containers owning widgets with the same labels) that are automatically identified in the interfaces and their underlying data models.

(4) *Nurture*: Using the forms they drew, the end-users then provide a set of positive and negative data samples, from which induction techniques allow to suggest possible constraints and dependencies that also need to be arbitrated. (5) *Bind*: The validated redundancies, constraints and dependencies are processed to perform an interactive integration of the key concepts elicited through the previous steps, hence leading to an integrated conceptual schema. (6) *Objectify*: A lightweight prototype application is generated from the integrated conceptual schema. It comprises a simple data manager that uses the interfaces drawn by the end-users and allows them to manipulate the concepts that have been expressed, typically to inspect, create, modify and remove data. (7) *Wander*: Finally, the end-users are invited to "play" with the prototype in order to ultimately validate the requirements, or identify remaining flaws.

In our doctoral research, we mainly focus on the five first steps, since the generation of the components is straightforward and the manipulation of a reactive prototype mainly adds another level of validation.

4 Discussion

In this section, we discuss the design and validation of our approach. More specifically, we ponder how the research context and the need for interactivity influenced the design of our approach, especially the tailoring or existing techniques, and we envision the obstacles threatening its assessment.

4.1 How to make end-users major stakeholders of the data requirements process?

The RAINBOW approach relies on the principles of the ReQuest framework [8, 9], which provides a complete methodology and a set of tools to deal with the analysis, development and maintenance of web applications. ReQuest deals with data modeling and the dynamic aspects of the future application, and proved that it is possible to efficiently and swiftly involve end-users in the definition of their needs. However, most laymen end-users were challenged by the task of designing dynamic and rich front-end interfaces supporting the business logic of their future application. Here, we therefore decided to focus specifically on improving the static data requirements process, leading the interfaces to appear as a means rather than an end product.

In particular, we wanted form-based interfaces to serve as a basis for discussion and joint development, and developed a tool to support this approach. To make the development of the interfaces more accessible and focus the drawing on the substance rather than (ironically) the form, we restricted the available graphical elements to the most commonly used ones and limited the layout of forms as a vertical sequence of elements [1], which also simplifies the mapping rules between the form model and the ER model. During the drawing, end-users must at least provide the label and cardinality of these elements, while advanced users may also provide integrity and existence constraints (which are normally

addressed during the following steps). The interfaces being drawn by non experts and possibly multiple end-users increases the possibility of inconsistencies among the labels used. In order to standardize the vocabulary from the start, we include a term analyzer that suggests alternative labels on-the-fly for new elements, using *String Metrics* [10] and the lexical reference system WordNet [11]. The same term analyzer is used during the *Investigate* phase to clarify any remaining ambiguity.

We mentioned in [1] the *structural redundancy* issue that we deal with during that same phase, and how our context led us to prefer a simple comparison algorithm to existing *frequent embedded subtrees mining* algorithms for rooted unordered trees. This case is very representative of the choices we made during the design of our approach. Take for instance the *Nurture* phase, during which we need to elicit integrity constraints (identifiers, cardinalities, value types and sizes, domain values, ...), existence constraints among optional fields (coexistence, at least one, exactly one, at most one, ...), and functional dependencies (when the values of certain fields may determine the values of other fields). Such constraints and dependencies can be discovered by analyzing existing data samples. Several techniques deal with this issue (candidate generate-and-test, minimal cover, ...), but they rely on large preexisting data sets. In our case, there is possibly no available data sample or the re-encoding would be too expensive, and it is anyways unrealistic to ask end-users to willingly provide numerous data samples.

These observations naturally called for new ways to discover and suggest constraints and dependencies on-the-fly, based on the incremental input of data samples by the end-users. Regarding the constraints, inductions can be made on these data samples to make suggestions on the value types (if all the instances of a given field are filled with numbers, this could suggest that the field is numeric or textual), the cardinalities (if all the instances of a given optional field are not empty, this could suggest that the field is actually mandatory), the existence constraints (when two optional fields are always filled at the same time, they could be coexistent), and so on...

As for functional dependencies (FDs), the ideal process should lead us to build a set of data samples and dependencies so that each entity type of the underlying conceptual schema becomes an Armstrong relation (i.e. a relation that satisfies each FD implied by a given set of FDs, but no FD that is not implied by that set). Reaching such a state is obviously not trivial, but we can try to near it progressively narrowing the FDs. We start by calculating the "strongest" valid candidates FDs for each entity type (i.e. each mandatory simple attribute of an entity type may determine the others), and maintain them until a data sample proves them wrong. Whenever a new data sample jeopardizes a functional dependency, the FD is discarded and valid alternative "lesser" FDs are recursively generated. To add interactivity, end-users can enforce or discard FDs by adding valid data samples for the entity type, arbitrating problematic data samples for a given FD (automatically generated from previously provided valid data samples), or even by directly enforcing or discarding FDs. This process should be repeated until there are only discarded and/or enforced FDs left for

each entity type. Possible identifiers should also be validated, knowing that an enforced FD $f : X \rightarrow Y$ may induce a identifier for the entity type E iff $X \cup Y = Attributes(E)$.

We consequently developed the RAINBOW tool kit, which is a user-oriented development environment, intended to assist end-users and analysts during the five first steps of our approach. The tool kit interacts with the repository of DB-Main, a database engineering CASE tool [12] providing all the necessary functionalities to support a complete database design process, as well as transformation tools and Database reverse engineering support. The interaction between these tools allows one to cover the whole database engineering process from both the end-user and the analyst perspectives.

4.2 How to validate the RAINBOW approach?

One of the most critical aspect of this research concerns its validation. The transversal nature of our approach, as well as the interdependence between the methodology and the tool support, naturally lead to the following research questions: (1) Does the RAINBOW approach and tool support help end-users and analysts to communicate (i.e. express, capture, validate) static data requirements to each other? (2) What is the quality of the conceptual schema produced using the RAINBOW approach?

The first question raises methodological (strategic design decisions), practical (potency and usability of the tool-support, added value for stakeholders) and sociological (end-user/analyst communication, empowerment, objectivity) issues, while the second question addresses the intrinsically complex predicament of quality assessment. Such problems are not easy to experiment, measure and validate, especially given the inherent difficulty of evaluating methodologies for the development of large systems : valuable learnings would require comparing our approach to existing ones, based on multiple experimentations led on numerous and different case studies over an extensive time span, which is not feasible at our level.

However, one of the contributions of our research is instead to define an experimentation canvas, based on preliminary studies that could in turn lead to a more realistic experimentation endeavor. The main idea to achieve this objective is to observe real-life implementations of our approach (using the *Participant-Observer* principles), then analyze the resulting conceptual schemas (using the *Brainstorming/Focus group* principles). Two independent studies have therefore been led, based on real-life issues concerning the two carefully chosen end-user participants EU1 and EU2. For each preliminary study, a pair of observers (including a main observer MO and a different assistant observer in each case) have therefore observed the interaction of one of the end-users with an analyst DB1 (the same in each case), jointly designing the conceptual schema of their dedicated project using the RAINBOW methodology and tool kit. Then, each resulting conceptual schema was discussed by three database analysts (DB1, DB2 and MO) to determine their qualities and flaws, as well as the delta between the "automatically" produced schemas and their "corrected" version.

Before starting the observation, EU1, EU2 and DB1 received a short training on the tool support and methodology based on screencast tutorials, and a session of questions/answers. The process was organized in four sequential sessions, each focusing on a peculiar aspect of the RAINBOW approach:

1. *Drawing the forms* (Represent): first of all, the end-users drew and edited the forms necessary to accomplish the tasks of their application project with the help of the analyst. They were asked to focus on the terminology and data aspect of this application, that is, the consistency of the labels and the specification of the widgets they needed, typically to input data.
2. *Analyzing the terminology of the forms* (Investigate): (1) the end-user and the analyst first analyzed the terminology of all the form elements to clarify any remaining ambiguities; (2) then, they analyzed the terminology of the containers to explain the relations existing between them. Whenever necessary, the pair went back and edited their forms.
3. *Providing examples and constraints* (Nurture): for each form, the pair first provided data samples then examined the technical constraints, the existence constraints, the functional dependencies and the possible identifiers associated with the form and its elements. Whenever necessary, the pair went back and replayed the previous steps.
4. *Finalizing the project* (Bind): from the previous steps, a set of "high level" concepts were materialized. For each of these concepts, the pair arbitrated the properties that were to be associated with the concept, then examined the associated technical constrains, the existence constraints, the functional dependencies and the possible identifiers. Whenever necessary, the pair went back and replayed the previous steps.

The observations and results of these preliminary studies must still be analyzed in depth, however they already raised several open questions, for instance regarding the drawing phase. During that step, the end-users naturally gave the commands to the analyst and were initially reluctant to manipulate the RAINBOW tool kit. On the other hand, the analyst did not feel very helpful or required for the process when he was not in charge of the drawing, though the end-users felt their presence reassuring. Who should therefore be drawing and who should be assisting? Is the drawing really a job for the end-user? What about the analyst's involvement and gratification?

We wanted to lead the end-user to focus on the content of the forms rather than their appearance, and subsequently chose an adaptative rendering for the widgets. For instance, selection widgets automatically switches from radio buttons to checkboxes or a selectable list according to the number of options and the cardinality of the field. However, this behavior surprised the end-users, and more generally they would have enjoyed at least a minimum of customization for the rendering of the widgets. Though the forms could be rendered afterwards in more stylish fashions (e.g. with HTML and CSS), could aesthetical considerations lead to a "bad" modeling, just because the end-users want the forms to be prettier? Can the analyst convince them that "it is ok even if it is ugly", and can the end-users really agree on that?

Likewise, the available widgets are restricted to forms, fieldsets, tables, inputs, selections and action buttons. For the two studies, these widgets seemed sufficient, although the composition sometimes called for creativity. We also observed that the end-users often drew single forms to collect multiple informations instead of drawing smaller, simpler forms (i.e. breaking the problems into smaller sub problems). Do the available widgets hence lead to the drawing of single oversized forms?

As previously mentioned, widgets have a cardinality specifying how many values could and should at least and at most be provided. We observed that the end-users often specified widgets as "mandatory", even if they sometimes acknowledged that it would not really be problematic if the given fields were not filled. Could the end-users therefore abusively use this type of cardinality while it is not really necessary? Do they understand the difference between a paper form, which can be submitted even if it is incorrect, and an electronic form which offers immediate acceptance or rejection?

As we can see, the drawing behavior of the end-users would make for an interesting research topic by itself, and so would the response of analysts to such an approach. These preliminary studies therefore highlight several sensible phenomenons that would require special attention on a larger scale experimentation. Besides, it would also be interesting to study the evolution aspects of our approach. While we already consider the possibility to "loop" during the steps of our approach as long as we are in the conceptual design, what would be the situation if we needed to edit a working database produced using our approach?

5 Conclusion

This paper presented a comprehensive interactive approach to bridge the gap between end-users and analysts during the conceptual modeling phase of database engineering. This approach supports the elicitation and validation of static data requirements with end-users, while overcoming several limitations of existing prototyping methods. It relies on the expressiveness and understandability of form-based user interfaces, used jointly with tailored Reverse engineering techniques to acquire data specifications from existing artifacts. Although our approach addresses a significant subset of these requirements, it does not cover all of its aspects, and therefore does not replace more traditional task and information analysis approaches, but rather complements them.

To get a better perspective on this research, we also discussed how the research context and the need for user involvement and interactivity influenced the design of our approach, especially the tailoring and combination of the existing techniques, as well as the inclusion of supportive tools.

Finally, we addressed the intrinsic difficulty to validate our transversal approach, while exposing a series of open questions raised by our ongoing preliminary studies. From the observations of these real-life implementations of our approach, we will define a set of guidelines for an experimentation canvas that could set the basis for a wider evaluation and an improved use of this approach.

Acknowledgments This doctoral research is being led under the supervision of Pr. Jean-Luc Hainaut, Laboratory of Database Application Engineering - PReCISE Research Center, Faculty of Computer Science, University of Namur, Belgium.

References

1. Ramdoyal, R., Cleve, A., Hainaut, J.L.: Reverse engineering user interfaces for interactive database conceptual analysis. In: Proc. CAISE'10 (to appear). (2010)
2. Rosson, M.B., Carroll, J.M.: Usability Engineering: Scenario-Based Development of Human-Computer Interaction. Morgan Kaufmann (October 2001)
3. Shoval, P., Shiran, S.: Entity-relationship and object-oriented data modeling-an experimental comparison of design quality. Data Knowl. Eng. **21**(3) (1997) 297–315
4. Terwilliger, J.F., Delcambre, L.M.L., Logan, J.: The user interface is the conceptual model. In: Proc. ER'06. Volume 4215 of LNCS., Springer (2006) 424–436
5. Hainaut, J.L.: The transformational approach to database engineering. In Lämmel, R., Saraiva, J., Visser, J., eds.: Generative and Transformational Techniques in Software Engineering. Volume 4143 of LNCS., Springer (2006) 95–143
6. Davis, A.M.: Operational prototyping: A new development approach. IEEE Softw. **9**(5) (1992) 70–78
7. Hall, P.A.V.: Software Reuse and Reverse Engineering in Practice. Chapman & Hall, Ltd. (1992)
8. Brogneaux, A.F., Ramdoyal, R., Vilz, J., Hainaut, J.L.: Deriving user-requirements from human-computer interfaces. In: Proc. IASTED'05. (2005) 77–82
9. Vilz, J., Brogneaux, A.F., Ramdoyal, R., Englebert, V., Hainaut, J.L.: Data conceptualisation for web-based data-centred application design. In: Proc. CAISE'06. LNCS, Springer (2006) 205–219
10. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. In: Proc. IJCAI IIWeb Workshop. (2003) 73–78
11. Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press (1998)
12. DB-Main: The official website `http://www.db-main.be`.