IADIS International Journal on Computer Science and Information Systems
Vol. 6, No. 2, pp. 1-23
ISSN: 1646-3692

INVOLVING END-USERS IN DATABASE DESIGN – THE RAINBOW APPROACH

Ravi Ramdoyal, Jean-Luc Hainaut
Laboratory of Database Application Engineering – PReCISE Research Center, Faculty of Computer Science, University of Namur, Rue Grandgagnage 21 – B-5000 Namur, Belgium

ABSTRACT

The first step of most information systems design methodologies consists in eliciting part of the user requirements from various sources such as user interviews and corporate documents. Regarding the core of the information system, that is, the database, these requirements are formalised into a conceptual schema of the application domain. Despite the intuitiveness and expressiveness qualities of conceptual formalisms, conceptual schemas have proved difficult to validate due to understandability limitations from the end-users standpoint. On the contrary, electronic forms are known to be a natural and intuitive way to express data requirements for laymen. Besides, the necessity to associate end-users of a future system with its specification and development steps has long been advocated. In this paper, we study data requirements elicitation techniques relying on user-drawn electronic forms. We explore the reverse engineering of form-based interfaces to perform an interactive database conceptual analysis, and subsequently present the tool-supported RAINBOW approach resulting from this investigation. This user-oriented approach relies on the adaptation and integration of principles and techniques coming from various fields of study, ranging from database forward and reverse engineering to prototyping and participatory design.

KEYWORDS

Information Systems, Requirements Engineering, Database Forward Engineering, Database Reverse Engineering, Human-Computer Interfaces, Prototyping, Participatory Design.

1. INTRODUCTION

In the realm of information systems engineering, requirements engineering is a key step that defines the necessary specifications for further analysis, design and development. Within this process, database engineering focuses on data modelling, where 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, large conceptual schemas have proved to be difficult to validate by laymen. In addition, end-users are often considered passive information sources, so that traditional database requirements elicitation techniques, such as the analysis of corporate documents and interviews of stakeholders, usually do not actively and interactively involve them in the overall specification and development process. Paradoxically, the necessity to associate end-users of a future information system within those steps has long been advocated. 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 analysts. Conversely, analysts should also be able to capture and validate these requirements by discussing them with end-users. Many users feel (and actually are) quite able to deal with complex data structures provided they are expressed through more natural and intuitive layouts, such as electronic forms.

This paper investigates the adaptation and integration of techniques coming from various fields of study, such as the reverse engineering of user-drawn form-based interfaces, to perform an interactive database conceptual analysis facilitating this communication and allowing users to be more deeply involved. Section 2 presents the research context and exposes the state of the art as well as the current limitations of available techniques. Section 3 presents the tool-supported RAINBOW approach for reverse engineering user-drawn form-based interfaces and how it deals with the different challenges put in light in the previous section. Section 4 discusses the specificities and contributions of this approach, while Section 5 addresses its validation. Finally, Section 6 concludes the discussion.

2. RESEARCH CONTEXT & STATE OF THE ART

Defining the application domain of an information system project and structuring the information that needs to be manipulated are the first steps of **Database Engineering**. This process of designing and implementing a database that has to meet specific user requirements has been described extensively in the literature (Batini et al, 1992; Elmasri & Navathe, 2006) and has been available for several decades in CASE tools. The most important and complex step of Database Engineering is the **Conceptual design** which aims at expressing user requirements into a conceptual schema, that is, a technology-independent abstract specification of the future database, also known as a Platform-Independent Model (PIM). From the conceptual schema, the transformational approach (Hainaut2006) allows database engineers to automate the production of logical and physical counterparts, targeting specific technology families and including performance-oriented aspects. Afterwards, from these schemas, well-mastered (semi-)automated techniques, which have long been studied in the database research community and applied in industry, allow artefacts of the final application to be produced: typically interfaces, programs, database code, etc. (Schewe & Thalheim, 2005).

Various techniques exist to elicit static data requirements, such as the analysis of corporate documents and interviews of stakeholders, and the Entity-Relationship (ER) model has long been the most popular medium to express such conceptual requirements (Shoval & Shiran, 1997). However, these techniques usually do not actively and interactively involve end-users, while the necessity to actively involve end-users of a future IT system during its specification and development steps has proved to be one of the key success factors. Besides, the ER formalism often fails to meet its objectives as an effective end-users communication medium, though its simplicity, its graphical representation and its availability in numerous CASE tools should make it the ideal communication medium between designers and users (Ramdoyal et al, 2010).

On the other hand, most users are quite able to deal with complex data structures, provided they are organized according to familiar layouts. In particular, electronic forms have proved to be more natural and intuitive than usual conceptual formalisms to express data requirements (Choobineh et al, 1992), while making the semantics of the underlying data understandable (Terwilliger et al, 2006). This strong link existing between graphical interfaces and data models is usually exploited in forward engineering, typically to produce artefacts such as form-based interfaces from a conceptual schema. Conversely, a form contains data structures that can be seen as a particular **view** of the conceptual schema.

The transition from one to another has been shown to be formally tractable (Rollison & Roberts, 1998), so that **Database Reverse Engineering** (DBRE) techniques can be applied to recover a fragment of the conceptual schema. Indeed, DBRE notably consists in recovering the database requirements (i.e. the conceptual schema) from multiple system artefacts that are usually obtained through schema transformation, such as documentation (when available), DDL code of the database, data instances, forms and source code of application programs (Chikofsky & Cross, 1990; Hall, 1992; Hainaut, 2002).

Such techniques can be combined with **Prototyping**, which usually acts as a basis for interviews or group elicitation (Nuseibeh & Easterbrook, 2000), while providing early feedback (Davis, 1992). A prototype can indeed by defined as a dynamic and interactive visual working model of user requirements that can be used as a communication tool for developers, customers and future end-users by providing the latter with a physical representation of key parts of the system before implementation (Connell & Shafer, 1995; Pomberger et al, 1991).

Deriving requirements from prototype artefacts has a long tradition. In 1984, Batini et al. studied paper forms as a means to collect and communicate data in the office environment (Batini et al, 1984). Later on, Choobineh et al. explored a form-based approach for database analysis and design, and developed an analyst-oriented Form Definition System and an Expert Database Design System that incrementally produced an ER diagram based on the analysis of a set of forms (Choobineh et al, 1992). Kösters et al. introduced a requirements analysis method combining user interface and domain analysis (Kösters et al, 1996), while Rollinson and Roberts studied the problem of non-expert customization of database user interfaces and developed a set of graph-oriented transformations to extract an Extended Entity-Relationship schema describing an interface's information content (Rollinson & Roberts, 1998). More recently, Terwilliger et al. defined the formal GUAVA (GUi As View) framework to use the user interface directly as a conceptual model, by exploiting the hierarchical nature of forms-based user interfaces to provide a simple representation of their informational content, including the relationships between forms (Terwilliger et al, 2006). Rode et al. investigated the feasibility of end-user web engineering for webmasters without programming experience and developed a prototypical tool for the end-user development of web application involving non professional programmers (Rode et al, 2005). Yang et al. also inquired about the WYSIWYG user-driven development of Data Driven Web Applications, while transparently generating their underlying application model on the fly (Yang et al, 2008).

We can observe that all these approaches rely on the same core principles: (1) build a set of form-based interfaces; (2) extract the underlying form model; (3) translate the form model into a working data schema; (4) progressively build an integrated data schema by looking for structural redundancies as well as constraints and dependencies. However, several limitations must be underlined in most of these approaches. First of all, end-users are not involved intimately in the overall process, and the tools provided for the drawing of the interfaces are not dedicated to this purpose and/or not convenient for end-users. The underlying form model of the interfaces must typically be constructed by analyzing the physical composition (layout) before the informational composition (content) of the form, and in parallel, the prototypical form-based interfaces do not use a generic language that would enable GUI generation of an application on any target platform. Regarding the coherence of the 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 ontological 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 the final integrated schema often lacks refinement, such as specialisation hierarchies, existence constraints or functional dependencies. Besides, this resulting schema is not systematically submitted to end-users in a way enabling easy validation, and its possible evolution through time is not considered. Lifting these limitations is clearly a necessity to design comprehensive interactive database design methodologies.

3. PROPOSAL

3.1 Challenges

As we have seen, Prototyping has proved to be an efficient technique to elicit and validate static data requirements. In particular, form-based interfaces appear to be a powerful means of communication between all the stakeholders of an Information System project, since they can be used to express formal data requirements and benefit from reverse engineering techniques to derive valuable data user requirements. However, prototypes are still mainly used as a one-way communication channel, since they are designed by analysts rather than end-users. Moreover, the limitations exposed in existing approaches call for a special attention to notably manage the unification of terminology and structure of the intended conceptual schema, its enrichment to include hierarchies, constraints and dependencies, as well as the generation of usable applicative components. Besides, since we also want to involve intimately end-users, we must also provide them with adequate means to express requirements and map them to their database engineering counterparts.

These challenges are dealt by specific disciplines, but their concerns and subsequent processing overlay in the context of our inquiry. **Database forward engineering** notably deals with the clarification of terminological and structural ambiguities, the elicitation of constraints and dependencies, schema integration and the generation of applicative components. For our purpose, **Database reverse engineering** can address the extraction of data schemas from form-based interfaces, while **Prototyping** should allow users to express and validate concepts and requirements through form-based interfaces. Finally, since we want to emphasize **user-involvement**, we need to find ways to involve them and possibly tailor and integrate existing techniques. From these observations, we can wonder if it could not be possible to make prototyping accessible to any of the stakeholders, in order to let them transparently express formal requirements on which could be applied transformational techniques, and therefore aim at an approach inspired by the principles of **Participatory Design** (Schuler & Namioka, 1993).

3.2 Overview

In order to bridge the gap between end-users and analysts to provide a better requirements acquisition process for Database Engineering and overcome the understandability limitations of the ER model, we propose to use user-drawn form-based interfaces as a two-way channel to express, capture and validate static data requirements with end-users by taking advantage of reverse engineering techniques.

More precisely, we consider an environment for which forms are a privileged way to exchange information and stakeholders are familiar with form-based (computer) interaction and the application domain. We claim that given such a context, we can exploit the expressiveness of form-based user interfaces and prototypes, and specialise and integrate standard techniques to help acquire and validate data specifications from existing artefacts in order to use form-based user interfaces as a two-way channel to communicate static data requirements between end-users and analysts. This interaction can produce a conceptual schema that includes integrity constraints, existence constraints and functional dependencies, and represents a major part of the application domain, which can be furthermore enriched in cooperation with other elicitation techniques.

Indeed, since existing artefacts can be used to recover the underlying requirements through well-mastered reverse engineering techniques, we advocate using such tailored techniques in forward engineering by working with the virtual artefacts produced by end-users. This approach benefits from the advantages of rapid prototyping, while making the user a central actor of the process, and designing a set of simple semantic interfaces rather than a complete application.

In order to formalise this approach, we need to take in account several specificities, among which a high level of interaction with end-users, the possibility to involve different levels of participants (ranging from laymen to experts) through a modular process, the need for a tool support accessible to end-users and useful to the analysts, as well as the necessity to tailor existing techniques. We indeed want to provide end-users with adequate tools to draw and specify by themselves the interfaces describing the underlying key concepts of their application domain, without having to worry about any application logic. Provided a little training and, as previously explained, involving end-users in such processes may have a very positive impact. In this context, the computer analysts rather appear as guides, whose roles are oriented towards the validation of requirements and the generation of complex code.

These principles are at the foundation of broader approaches, such as the ReQuest framework (Vilz et al, 2006), 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 alternative **RAINBOW approach** keeps the same overall philosophy while focusing on the specification of static data requirements as part of a greater Requirements Engineering process. The specificities of this approach led us to specialise and integrate existing techniques into a semi-automatic seven-step process (see Figure 1) that does not aim to provide a ready-to-use application, but a set of specification documents and tools, in order to support the development of future applications and overcome the previously mentioned limitations while dealing with the cited challenges.

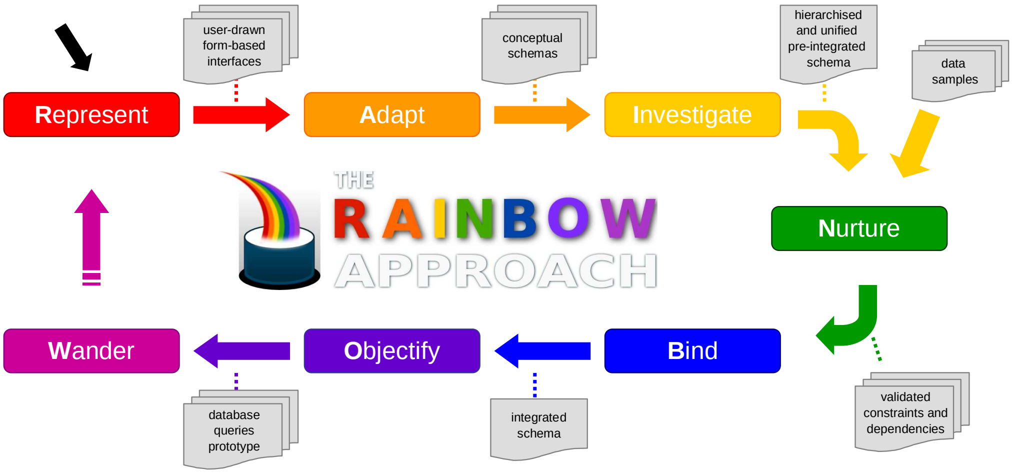

user-drawn form-based interfaces
conceptual schemas
hierarchised and unified pre-integrated schema
data samples
Represent
Adapt
Investigate
Nurture
THE RAINBOW APPROACH
Wander
Objectify
Bind
validated constraints and dependencies
integrated schema
database queries prototype

Figure 1. Overview of the RAINBOW approach.

Note that in the scope of this research, we work with the Generic Entity-Relationship (GER) model (Hainaut, 1989), which is an extended Entity-Relationship model that includes, among others, the concepts of schema, entity type, generalisation hierarchy, relationship type of any degree, value domain, entity type and relationship type attribute (including compound and/or multivalued attributes), key, as well as various constraints. The GER is a wide-spectrum model that encompasses several abstraction levels (from conceptual to physical) and most modelling paradigms (e.g., ER, object-relational or XML)

Let us now present for each of the RAINBOW steps its overall objective, the issues it handles, the existing solutions to manage these issues and why they are not suitable in this context, as well as how they have been adapted and integrated for our purpose. More detail, as well as the algorithms formalising the presented strategies, can be found in (Ramdoyal, 2010).

3.3 Represent: Enabling users to express Concepts and Requirements by building themselves Form-Based Interfaces

In the first step of the approach, end-users are invited to draw and specify 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. The end-users must at least provide basic properties regarding the interface elements (typically a label and description). Advanced users may also provide other properties such as the size of a field, the expected type of values, default or predefined values, existence constraints, as well as links between the concepts. Note that the objective is **not** to let end-users draw the interfaces of a future application, but to capture requirements through a medium they are familiar with.

Numerous User Interface Description Languages (UIDL) exist to model form-based interfaces, however, since they must express rich and complex interfaces, layouts and behaviours, their structure becomes complex and difficult to read, and furthermore, end-users may be overwhelmed by their superabundance of available widgets and compositions. In order to focus on simple interface widgets that can allow end-users to simply express concepts, while casting away the technical aspects of layout, we propose a simplified form model based on the most primitive and usual form widgets (Figure 2), which can intuitively be mapped to the GER model and through which any other widget can be expressed. These widgets are either containers such as `forms`, `tables` and `fieldsets` or simple widgets such as `inputs`, `selections` and `buttons`.

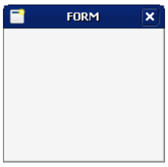
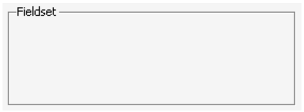
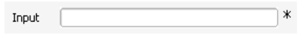
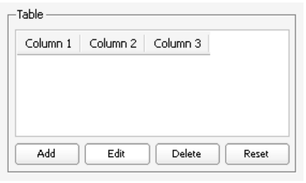
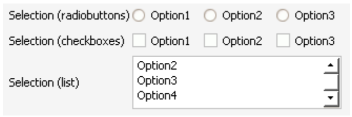
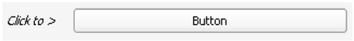
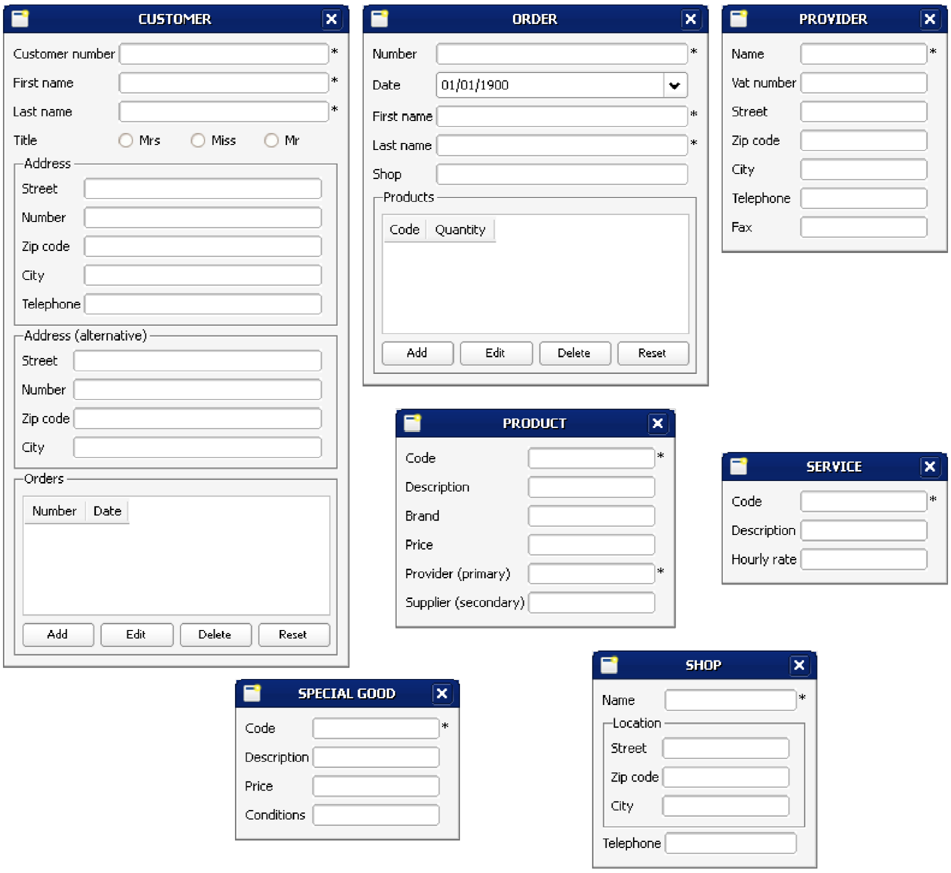
Widget	ER Counterpart	Widget	ER Counterpart
	Entity Type		Monovalued Compound Attribute
			Mandatory Monovalued Simple Attribute
 Table Column 1 Column 2 Column 3 Add Edit Delete Reset	Multivalued Compound Attribute	 Selection (radiobuttons) ○ Option1 ○ Option2 ○ Option3 Selection (checkboxes) □ Option1 □ Option2 □ Option3 Selection (list) Option2 Option3 Option4	Mono/Multi-valued Simple Attribute with Value Domain
			Procedural Unit

Figure 2. Widgets of the RAINBOW Simplified Form Model

A dedicated tool support has been developed to manipulate this model which is intended to be transparently used by end-users to express concepts, and includes information for designers and CASE developers that would like to instantiate or extend it. Figure 3 illustrates the type of interfaces that could be produced during this phase if one, for example, wanted to support the management of a small company that offers services and sales products. For instance, for each customer, personal information including his main and alternative addresses is stored, as well as the list of orders that he issued. Each of these orders mentions information on the context of its creation, and list the associated list of products, and so on.

3.4 Adapt: Inferring Data Schemas from Form-Based Interfaces

Once the interfaces are drawn, database reverse engineering techniques are applied to recover the underlying conceptual schema of the domain. The interfaces are automatically analyzed to extract data schemas using mapping rules, a subset of which is presented in Figure 2. Then, each individual entity type is transformed into a primitive conceptual schema by transforming complex attributes into entity types. Figure 4 illustrates the expected type of conceptual schemas obtained from the translation of user-drawn form-based interfaces.



CUSTOMER

Customer number *

First name *

Last name *

Title ○ Mrs ○ Miss ○ Mr

Address: Street, Number, Zip code, City, Telephone

Address (alternative): Street, Number, Zip code, City

Orders

Number	Date

Add | Edit | Delete | Reset

ORDER

Number *

Date 01/01/1900

First name *

Last name *

Shop

Products

Code	Quantity

Add | Edit | Delete | Reset

PROVIDER

Name *

Vat number

Street

Zip code

City

Telephone

Fax

PRODUCT

Code *

Description

Brand

Price

Provider (primary) *

Supplier (secondary)

SERVICE

Code *

Description

Hourly rate

SPECIAL GOOD

Code *

Description

Price

Conditions

SHOP

Name *

Location: Street, Zip code, City

Telephone

Figure 3. Possible user-drawn form-based interfaces for the management of a small company that offers services and sales products

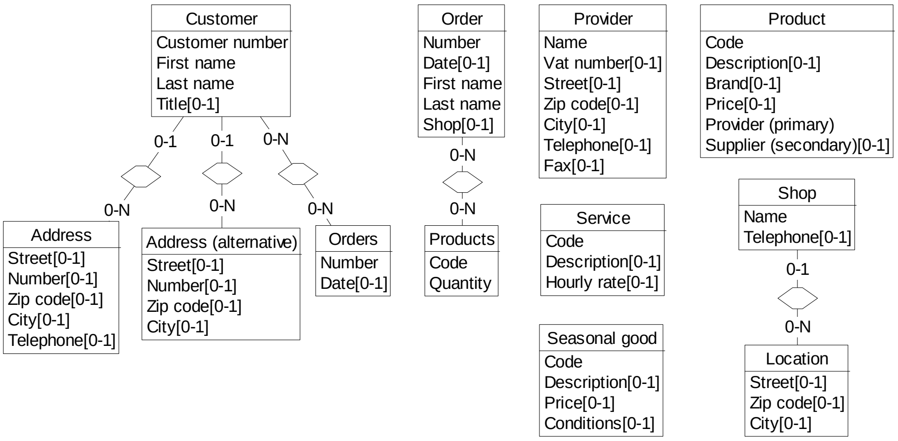


Figure 4. The conceptual schemas corresponding to the interfaces of Figure 3.

3.5 Investigate: Analysing Semantic and Structural Redundancies to Manage Commonality

Cross-analysing the set of individual schemas obtained by adapting the user-drawn forms usually brings to light possible ambiguities as well as redundant information. Typically, whereas we can observe that the constructs used by a single user are relatively consistent among the interfaces he draws, when considering multiple users, we notice that variabilities, ambiguities and redundancies may occur. These phenomena may concern the various properties of the widgets, such as the label, description, minimal cardinality, maximal cardinality, value type, value size, or domain of values... In the scope of this paper, we focus on the widgets labels to track down semantic and structural ambiguities, but the definitions and strategies that we propose could intuitively be extended to take in account other properties.

3.5.1 Terminological Ambiguities

Terminological ambiguities occur when elements of the schema are **semantically similar**, i.e. their names appear to be orthographically and/or ontologically similar, which respectively concerns their spelling and meaning. In the example, the labels “Orders”' and “Order” are orthographically similar, while the labels “Provider” and “Supplier” are ontologically similar. Identifying orthographically similar strings is a problem usually dealt with by using **String Metrics** (Cohen et al, 2003). **Ontologies, Thesaurus** and **Dictionaries** can also be useful to track down similarities of meaning among a set of words, and may target specific domains, e.g. UMLS for the medical field (Hersh et al, 2000).

In order to discover such similarities, we compare the labels of each interface using a variant of Jaro-Winkler's distance (Winkler, 1990) and WordNet (Fellbaum, 1998). The former is one of the most popular string metrics for dealing with word comparison, while the latter is an English non domain-specific orthographical reference system, handling nouns, verbs, adjectives and adverbs, and providing definitions, synonyms and hypernyms.

The discovered similarities are then highlighted in the interfaces and submitted to the end-users for arbitration. This task consists in deciding which semantic similarities are actually genuine **semantic equivalences**, i.e. the similarities that are agreed upon by the end-users and the analysts to represent the same concept. For each equivalence, a unifying term is defined and propagated to the schemas and the forms.

3.5.2 Structural Redundancy

The second type of similarity that may occur is the **structural redundancy**. Typically, we can observe that entity types can share components (attributes and roles) bearing the same names, which suggests that these elements may induce different degrees of similarity.

Given the tree-like structure of the conceptual schemas, the problem of mining structural redundancies is actually alike the problem of **tree mining** (Chi et al, 2005), and more precisely **frequent embedded subtrees mining in rooted unordered trees** (Jimenezet al, 2008). Tree based approaches are suitable for complex and deep graphs, however we observe that the structure of user-drawn interfaces is usually quite simple (with rarely more that three or four levels of imbrication), if only by concerns of legibility and usability. Indeed, “most forms have a shallow (i.e. few levels) and narrow (few nodes per level) structure because of human information processing limitations” (Choobineh et al, 1992).

Instead of putting in motion such heavy algorithms, we therefore propose to adopt a simpler strategy that consists in comparing one by one the entity types to outline patterns, i.e. bijections between two sets of components belonging to different entity types. The similarity between components from each set is measured using several indicators (typically, the label). The discovered patterns are highlighted in the forms, and the end-users are then invited to arbitrate them by classifying the relation between the concepts sharing a pattern among one of these most usual cases, as illustrated in Figure 5: (a) **difference** (the entity types fortuitously share a set of components), (b) **equality** (the entity types represent the same concept), (c) **union** (the entity types partially represent the same concept, which may translate into the specialization of a higher-level concept non explicitly expressed), (d) **comprehension** (one of the entity types is a specialization of the other), (e) **complementarity** (one of the entity types actually refers to the other). As for the terminological arbitration, unifying terms are defined and propagated to the schemas and the forms.

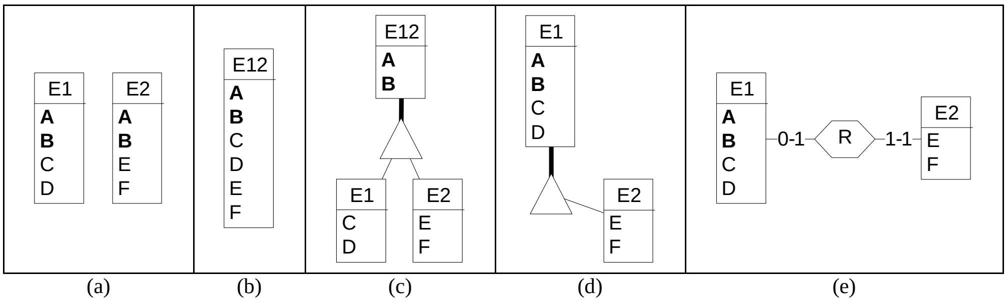


Figure 5. Most common cases of structural redundancy

At the end of this interactive process, we obtain a pre-integrated schema resulting from the terminological and structural analysis of the set of schemas obtained through the Adapt phase. In this schema, the terminology has been unified so that every element associated with a given term now represents the same concept. Also, the sub-schemas originally associated with each form are now connected through the relationship types and IS-A hierarchies of their entity types, as illustrated in Figure 6.

3.6 Nurture: Eliciting Dependencies and Constraints

In order to enrich the pre-integrated schema, we then focus on uncovering additional constraints and dependencies on its elements. Though these constraints can be provided directly, it appears that the acquisition and use of data samples may also be useful and more natural in this process. Indeed, not only do data samples test the ability of the user-drawn form-based interfaces to gather the necessary information, but they also help to visualise the implications of existing constraints. Moreover, their analysis may in turn reveal possible unsuspected constraints. Using the interfaces they drew, end-users are therefore invited to provide data examples that are analysed to infer and arbitrate possible constraints and dependencies.

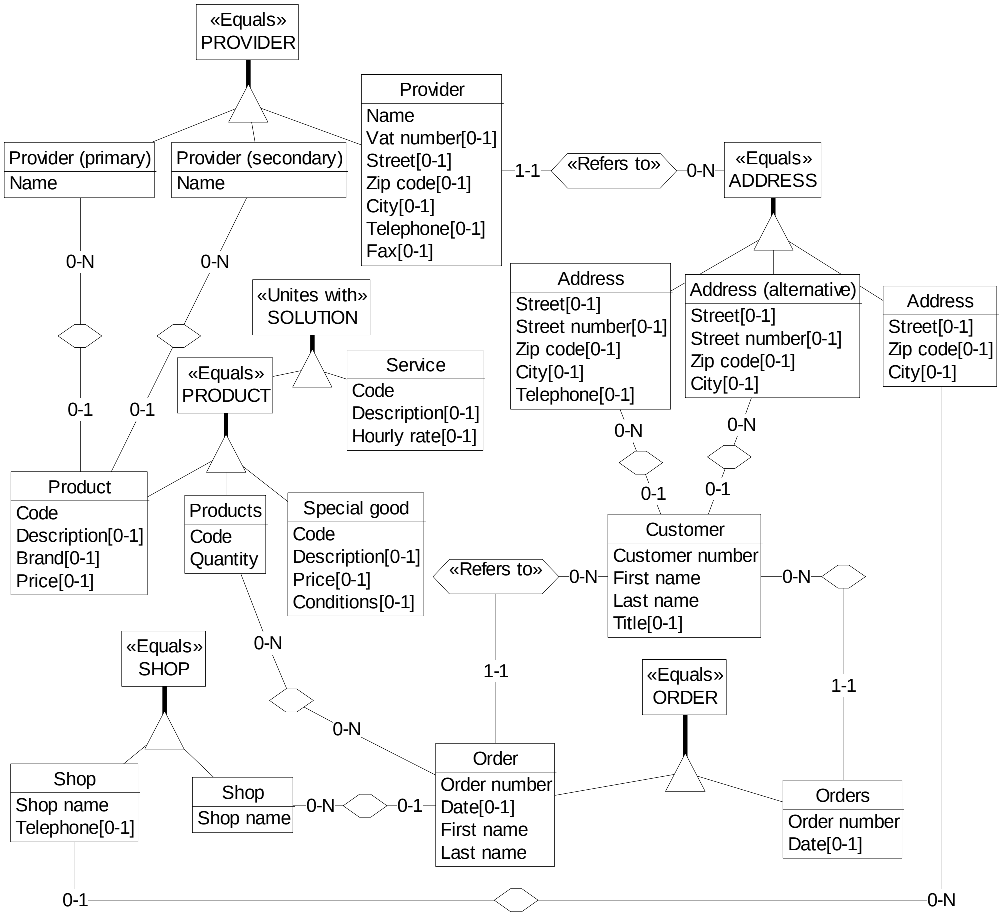


Figure 6. The pre-integrated schema of the example. The newly created supertypes and relationship types are marked with stereotypes expressing their meaning.

There are numerous types of constraints and dependencies that can be established for a given schema, but in this research, we especially focus on:

- **Technical constraints:** minimal and maximal cardinalities, value type, value size, prerequisite optional components (for optional components);
- **Existence constraints**, which define how the optional components should coincide for each entity type (coexistence, exclusive, at-least-one, at-most-one);
- **Functional dependencies**, which define the implications between sets of components;
- **Identifiers**, which define the sets of components that uniquely identify a given instance of a given entity type.

Analysing the content of a database or a subset of data samples and using **induction** can intuitively lead to make assumptions on possible technical constraints, existence constraints and identifiers. Consider for instance an optional textual attribute A. If for all the data samples provided so far, we observe that the widget associated with A is never empty and always composed of a number, we could easily wonder if A is not actually a mandatory numeric attribute. Moreover, if all the values provided for that widget are different, this could suggest that A is in fact a primary or secondary identifier.

As for discovering functional dependencies, this problem is usually dealt by using **dependency discovery problem** approaches, based on the analysis of a given database content, such as DepMiner (Lopes et al, 2000) or FD_Mine (Yao & Hamilton, 2008). Unfortunately, we observed that the existing approaches rely on massive pre-existing data sets, which is here problematic. Indeed, given our context, there is possibly no available data sample, or their re-encoding would be too expensive. It is anyway unrealistic to ask end-users to willingly provide numerous data samples. This naturally calls for new ways to discover and suggest constraints and dependencies on-the-fly, based on the incremental input of data samples by end-users. We therefore propose an interactive process inspired by the principles of **Armstrong relations** (Lopes et al, 2000), and enabling end-users to provide such constraints and dependencies as well as data samples in whatever order.

An Armstrong relation is a relation that satisfies each functional dependency implied by a given set of functional dependencies, but no functional dependency that is not implied by that set. 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 such a relation. However, reaching such a state is obviously not trivial per se, and these principles are here inapplicable as a side effect of user involvement. However, we can try to near it by progressively narrowing the functional dependencies.

The process therefore starts with the initialisation of the constraints and dependencies based on previously provided requirements, then relies on user input to gather data samples and constraints. Since the number of possible functional dependencies for each entity types can be very high, we prefer to initialise a set of high-level possible dependencies, which would be the most general yet restrictive ones. These high-level dependencies claim that any component of a given entity type could determine the combined values of the other components. Whenever a dependency is dismissed or invalidated, we recursively generate weaker functional dependencies to cover all the existing ones, by progressively reducing the right-hand sides and enlarging the left-hand sides. The objective is to favour functional dependencies with minimal left-hand sides and maximal right-hand sides.

When a new data sample is added, we analyse each valid functional dependency to check if there is an existing data sample that is conflictual, i.e. if an existing data sample has the same left-hand side but a different right-hand side when considering the components of the functional dependency. If such a conflictual data sample exists, the functional dependency is discarded and alternatives are recursively generated. End-users can also directly specify enforced or discarded constraints and dependencies, even without looking at possible suggestions. To be accepted as enforced, a given constraint or dependency must be satisfied by the existing set of data samples associated with the considered entity type. On the other hand, it can be discarded as long as it still qualifies as a constraint or dependency for the given entity type.

Acquiring data samples hence enables us to generate suggestions for possibly valid constraints and dependencies, while acknowledging constraints and dependencies restricts the data samples that the users may provide. This interaction enables to progressively narrow the set of possible constraints and dependencies, while building a set of valid data samples that will prove useful later on. Once this interactive process ends (hopefully, with all the constraints either validated or rejected), we accordingly proceed with the update of the pre-integrated schema. During the overall process, the role of the analyst is crucial to guide end-users and ensure that no relevant constraint or dependency is ignored.

3.7 Bind: Completing the Integration of the Conceptual Schema

We subsequently address the final integration of the elements of the pre-integrated schema based on all the previously collected specifications. Different transformational techniques exist to handle the integration of similar objects into non-redundant structures (Spaccapietra et al, 1992). Among these techniques, we choose to work with **n-ary integration** for handling upward inheritance and solving the constraints, because of the potential multiple occurrences of key concepts, and with **binary integration** for referential components and attributes that need to be moved from entity types into relationship types.

In particular, we analyse and refine the IS-A hierarchies resulting from the Investigate step in order to elicit components that can be integrated and inherited upwardly. Referential attributes are moved and integrated into the appropriate entity types, as well as attributed describing relationships rather than their current entity type owner. Finally, the constraints and dependencies from the Nurture step have been updated accordingly. Since these tasks can be led in any order, the role of the analyst is crucial to manage the process and guide end-users appropriately. Besides, his skills may be needed to refine the structure of the schema once the previous tasks are completed, in order to ensure the validity of the schema.

At the end of this step, the integrated schema therefore crystallises all the requirements that were expressed by end-users since the beginning of the RAINBOW process, as illustrated in Figure 7. The schema should hence represent the application domain as expressed though the user-draw forms and the additional specifications that were provided. The aim of the remaining steps is to ultimately confirm these requirements.

3.8 Objectify: Generating Applicative Components

A lightweight prototype application is then 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. Such a process is relatively straightforward (Elmasri & Navathe, 2006).

First of all, a database can be automatically generated using the transformational approach: the integrated conceptual schema is sequentially transformed into a logical schema, then a physical schema, and finally DDL code, from which an operational database can be created using a compatible Database Management System (DBMS). CASE tools have proved very effective in supporting such a process. Subsequently, access keys, table spaces and clusters can be generated. Afterwards, if judged relevant by the participants, the database can be populated with the data samples provided by the end-users. Once the database has been set up in the DBMS, simple queries SQL to select, insert, update and delete rows of each table can be automatically generated. These queries can subsequently be connected to the form-based interfaces drawn by the end-users in order to make them reactive and report the messages of the database.

The final step consists in grouping the user-drawn interfaces in an operational environment. This implies creating the mechanisms for a central application granting navigational access to the forms and between them. The prototypical application thus created makes it possible to perform simple consulting and editing actions on the database through the form-based interfaces, which would qualify it as a lightweight data manager for the database.

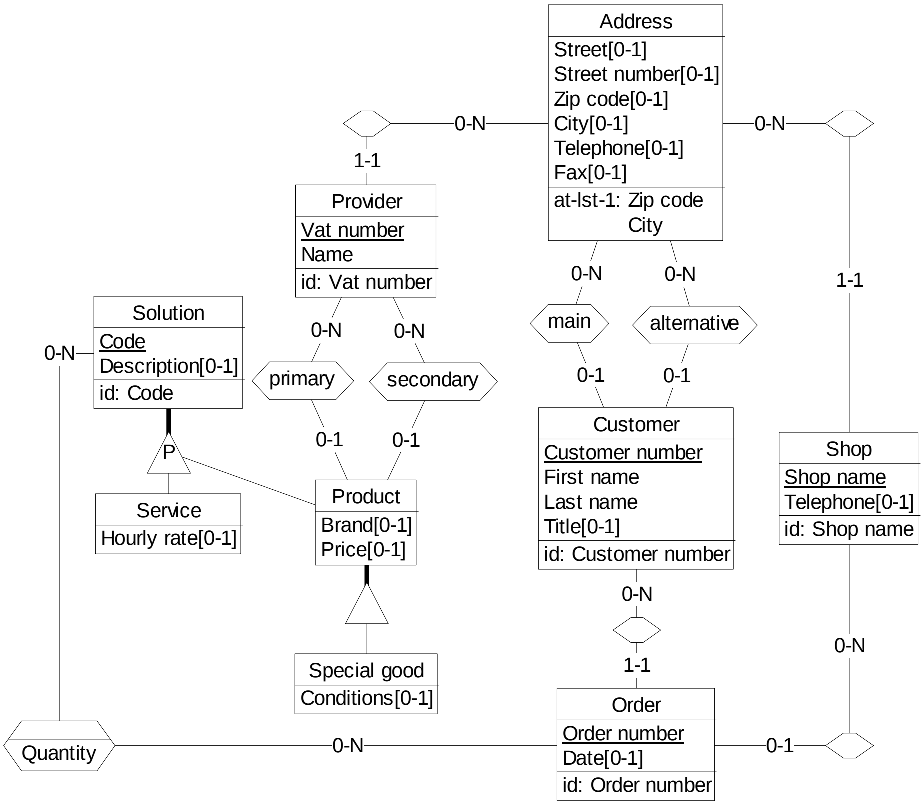


Figure 7. The final integrated schema of the example.

3.9 Wander: Validating the Requirements through the generated Prototype

Finally, the last stage consists in confronting the end-users with the prototypical application to check if the static data requirements that were materialised meet their needs, which should ultimately validate the integrated conceptual schema. The role of the analyst during this process is therefore to assist the users in the validation of the schema through the use of the prototype, and to record their positive and negative remarks. The evaluation of the elicited requirements through the manipulation of the associated lightweight data manager should eventually lead to end the requirements elicitation process or to loop back to the previous steps to add, delete or modify the specifications that were expressed.

3.10 Tool Support

In order to assist the end-users and analysts during the different steps of this interactive approach, a dedicated tool support has been developed. The RAINBOW Tool Kit is a user-oriented development environment, intended to assist end-users and analysts in the definition and validation of database requirements through prototyping.

It supports the first steps of the approach, starting with the initial drawing step. For this purpose, the tool kit provides ready-to-use implementations of the widgets presented in Figure 2, in order to simplify the elaboration of form-based interfaces. Figure 8 shows for instance the edition of the forms presented in Figure 3.

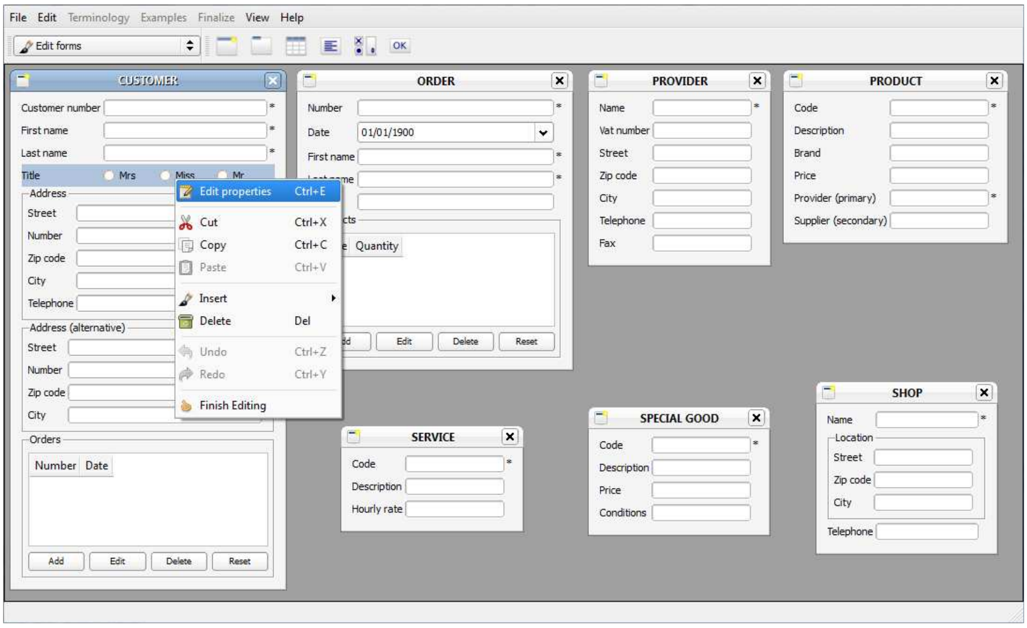
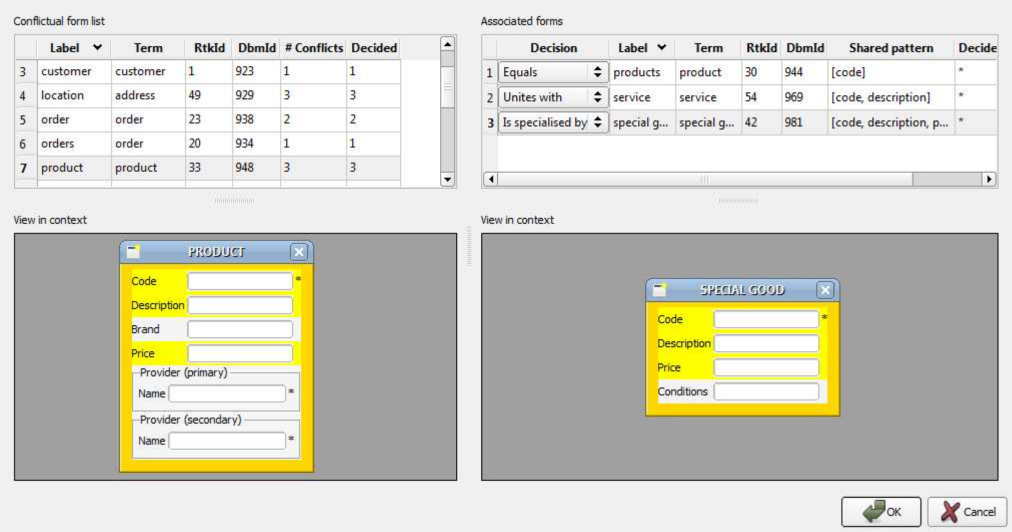


Figure 8. Editing of the forms of Figure 3, using the RAINBOW Tool Kit.

Once the drawing is completed, the mapping rules of Section 3.4 are automatically applied to create the underlying data models. To proceed with the unification of the terminology and structure, the forms are automatically analysed and the elicited ambiguities are highlighted in the interfaces so that they can be arbitrated, as illustrated in Figure 9.

Once the arbitration of these ambiguities has been performed, the users proceed with the interactive elicitation of constraints. For this purpose, they can directly provide data samples or constraints for each form, or validate candidate suggestions, as illustrated in Figure 10. Each new data sample restricts the set of candidate constraints that have not been validated yet, and conversely, each new validated constraint restricts the values that can be provided for subsequent data samples. Finally, mechanisms are provided to handle the final integration of the underlying data models.



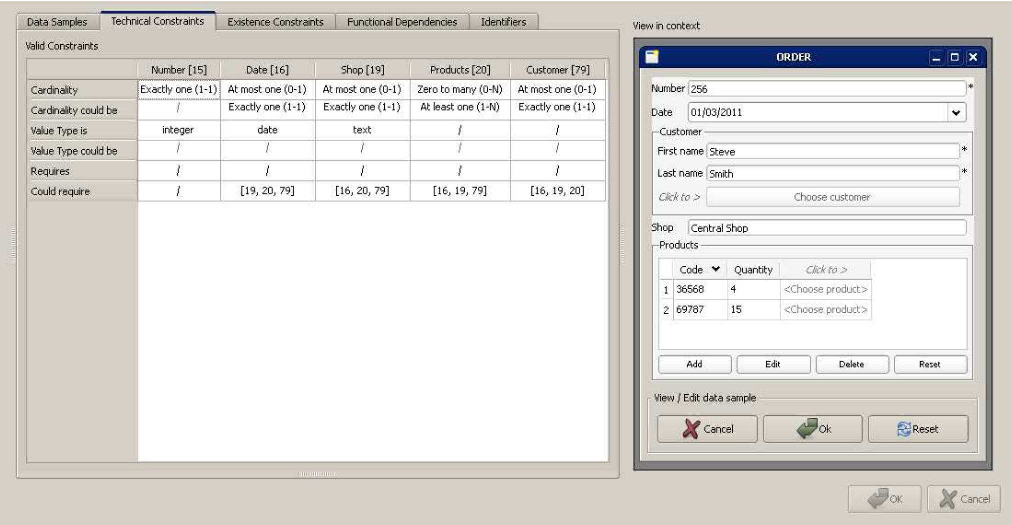
Conflictual form list

	Label	Term	RtkId	DbmId	# Conflicts	Decided
3	customer	customer	1	923	1	1
4	location	address	49	929	3	3
5	order	order	23	938	2	2
6	orders	order	20	934	1	1
7	product	product	33	948	3	3

Associated forms

	Decision	Label	Term	RtkId	DbmId	Shared pattern	Decide
1	Equals	products	product	30	944	[code]	*
2	Unites with	service	service	54	969	[code, description]	*
3	Is specialised by	special g...	special g...	42	981	[code, description, p...	*

Figure 9. Arbitrating the structural similarity `{Code,Description,Price}` existing between the forms `Product` and `Special Good`, using the RAINBOW Tool Kit.



Valid Constraints

	Number [15]	Date [16]	Shop [19]	Products [20]	Customer [79]
Cardinality	Exactly one (1-1)	At most one (0-1)	At most one (0-1)	Zero to many (0-N)	At most one (0-1)
Cardinality could be	/	Exactly one (1-1)	Exactly one (1-1)	At least one (1-N)	Exactly one (1-1)
Value Type is	integer	date	text	/	/
Value Type could be	/	/	/	/	/
Requires	/	/	/	/	/
Could require	/	[19, 20, 79]	[16, 20, 79]	[16, 19, 79]	[16, 19, 20]

Number 256 *
Date 01/03/2011
Customer
First name Steve *
Last name Smith *
Click to >
Shop Central Shop
Products

	Code	Quantity	Click to >
1	36568	4	<Choose product>
2	69787	15	<Choose product>

Figure 10. Providing data samples and constraints for the form `Order`, using the RAINBOW Tool Kit.

The tool kit is written in Java, using QT Jambi for the rendering of the interfaces, as well existing libraries to manage Jaro-Winkler's distance and interact with WordNet. It also interacts with the repository of DB-Main, a database engineering CASE Tool providing all the necessary functionalities to support a complete database design process (from conceptual analysis to DDL code generation). More technical detail on the implementation of the tool kit can be found in (Ramdoyal, 2010).

4. SPECIFIC ASPECTS OF THE RAINBOW APPROACH

4.1 Integrating Different Disciplines

The RAINBOW approach is at the crossroad of different disciplines, each of which deals with specific issues using dedicated methods and techniques. However, their concerns and subsequent processing can concur for the purpose of bridging the gap between end-users and analysts in order to elicit static data requirements. One of the main achievements of this research was therefore to identify, tailor and integrate principles and techniques coming from the fields of Database Forward Engineering, Database Reverse Engineering, Prototyping and Participatory Design in order to provide this interactive and user-oriented Database Conceptual Analysis approach. The following specificities naturally follow from the decisions that were made to support this integration into a consistent and comprehensive approach, for which the contributions, with respect to the limitations of existing approaches, are synthesized in Table 1.

4.2 End-users as Major Stakeholders throughout the Data Requirements Process

In this research, the focus was specifically put on simplifying and 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, hence using prototyping in an exploratory fashion, though it could also be used in an evolutionary approach. Several challenges inherent to this user-centred approach had to be therefore managed.

First of all, to make the development of the interfaces more accessible and to focus the drawing on the substance rather than (ironically) the form, the available graphical elements were restricted to the most commonly used ones, which incidentally also simplifies the mapping rules between the form model and the ER model, and a dedicated tool was designed to support this process. The possible lexical variations that could occur in such an interactive process are taken in account, while simply ignored by other similar researches. We therefore offer the possibility to detect and correct on-the-fly many mistakes or deviations in the terminology, or to deal with them later on.

Besides, the interfaces are systematically used to visualise similarities, to input constraints and data samples, so that they can be the referent for end-users, and their favourite communication means. The end-users therefore interact with the form-based interfaces, while the analyst can also access and edit the underlying data schemas at any time, as long as he ensures the maintenance of the mapping.

The will to involve intimately end-users into the definition of their needs and the specification of the static data requirements, while managing the satisfaction of all the stakeholders, also places this approach as more suitable for information systems projects in small to medium size enterprises. Besides, the projects should be themselves small to medium sized, in order to maintain a manageable set of form-based interfaces.

Method	**FDS/EDDS**	**FLUID**	**/**	**Click**	**GUAVA**	**AppForge**	**RAINBOW**
Authors	Choobineh et al, 1992	Kösters et al, 1996	Rollinson & Roberts, 1998	Rode et al, 2005	Terwilliger et al, 2006	Yang et al, 2008	Ramdoyal et al, 2010
Tool support (drawing and analysis)	Form Definition System + Expert Database Design System	DIWA + Extended PCTE object management system	Xfig + Prolog + GRL + XVCG	Click	GUAVA framework	AppForge	Rainbow Tool Kit
Prototyping finality	Exploratory, evolutionary	Evolutionary	Exploratory, evolutionary	Evolutionary	Exploratory	Evolutionary	Exploratory, (evolutionary)
Prototype designers	Analysts, end-users	Analysts	Analysts	Analysts, end-users	Analysts	Analysts, end-users	Analysts, end-users
Underlying form model	/	User Interface Analysis + Object (UIA /UIO) models	TRIDENT variant	HTML/PHP	GUAVA-tree	HTML	RSFM
Syntactic schema matching	/	/	plurals	/	/	/	orthographic, ontological
Structural schema matching	equality	equality	equality	/	/	equality	equality, specialisation, union, complementarity
Constraints and dependencies	identifiers, FDs	identifiers	identifiers	identifiers	identifiers	identifiers	existence constraints, identifiers, FDs
Examples analysis	static, user-provided and/or generated	dynamic, user-provided	/	/	/	/	static, user-provided and/or generated
Data model	ER	OOA	ERC+ (EER)	Relational Model (MySQL)	Relational model (Natural schema)	ER	GER
Model life cyle	linear	linear	linear	linear	linear	linear	cyclic
Target platform	Unrestricted	Unrestricted	Unrestricted	Web-oriented	Unrestricted	Web-oriented	Unrestricted

Table 1 - Comparison of existing approaches in prototypical reverse engineering for forward engineering. The symbol “/" means that no details were explicitly provided for the given characteristic.

It is interesting to note that though the approach is oriented towards end-users, the real corner-stone of the RAINBOW processes is the analyst. Indeed, his social and technical skills and knowledge are crucial to manage, assist and guide end-users in order to perform an enjoyable and effective elicitation process for all the parties involved.

4.3 Using Reverse Engineering for the Purpose of Forward Engineering

Reverse engineering usually 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. However, using controlled artefacts and monitored processes, the objective is here to “build the truth” rather than “find the truth”. In particular, the form-based interfaces are used as a well-defined specification language, as opposed to the usual reverse engineering approach, where the existing screens are obscure artefacts that need to be decrypted. This requires to significantly adapt the usual Database Reverse Engineering (DBRE) methodology (Hainaut2002).

DBRE typically comprises the following four sub-processes: (1) **Physical extraction**, which consists in parsing the DDL code in order to extract the raw physical schema of the database; (2) **Refinement**, which enriches the raw physical schema with additional constructs and constraints elicited through the analysis of the application programs and other sources; (3) **Cleaning**, which removes the physical constructs (such as indexes) for producing the logical schema; (4) **Conceptualisation**, which aims at deriving the conceptual schema that the logical schema implements. Such a methodology is obviously not applicable as is in the context of the of RAINBOW approach.

Indeed, starting from a set of user interfaces, the physical extraction does not allow one to derive a complete physical schema, but a set of **partial views** of this schema. Similarly, the refinement process may not rely on additional available artefacts such as application programs or database contents. However, it can take benefit from data samples provided by the users through the interfaces they have drawn, leading to the identification, among others, of candidate dependency constraints and attribute domains. The recovered constraints, once validated, are used to enrich the physical schemas in order to obtain a set of logical schemas. The cleaning phase, as defined above, does not make sense in the absence of an initial DDL code. Instead, the conceptualisation step allows one to derive a set of partial conceptual schemas from the logical schemas obtained so far. In particular, the logical schemas are normalised in order to ease the identification of similarities between them. This important process relies on transformation techniques. During the integration phase, the partial conceptual schemas are merged, based on structural and semantic similarity criteria, in order to produce a single complete conceptual schema.

4.4 A Modular and Non Standard View Integration Process

One of the key assets of this approach is its flexibility, especially regarding the enrichment of the data schemas. As we have seen, proficient end-users can already provide constraints during the drawing phase. Otherwise, such properties can be directly provided later on, or discovered from a set of data samples provided by end-users. Similarly, the unification of the terminology and structures can also be led during the drawing phase, or during further steps.

This modularity makes the approach suitable for different types of users, ranging from the layman end-user to the advanced Database Engineering, or from the analyst to the developer. The progressive gathering of elements of integration for further resolution also differs from the standard integration processes.

4.5 System Evolution Support through a Transformational Approach

The RAINBOW approach also heavily relies on the transformational paradigm, according to which most (if not all) Database Engineering processes can be modelled as a chain of schema transformations (Hick & Hainaut, 2006; Hainaut et al, 2008). The transformations that we use are incremental and preserve the semantics of source constructs in their target counterpart, which ensures the consistency, traceability and reversibility of the specified elements throughout the whole workflow. This also favours the evolvability of the specifications produced via the approach. Indeed, this approach is designed to loop if necessary, while storing and propagating all the previously provided specifications and decisions. Combined with the traceability of the elements, we can ensure the propagation of any modification in the different steps of this approach.

4.6 A Rich and Relevant Part of Requirement Specifications

The output of this process is a set of annotated form-based interfaces and their underlying integrated conceptual schema, as well as their associated playable prototype and ready-to-use database. Compared to other existing approaches, the resulting conceptual schema is rather rich, since it includes hierarchies, as well as constraints and dependencies. It can also be analysed to generate a thesaurus of the application domain. Moreover, the elements produced can effectively be used to share and validate requirements as part of the requirement specifications for a complete information system project, built around the database. Indeed, the RAINBOW approach ensures their validation and correction, and these artefacts can be used for further evaluation and reference, while contributing to the forecast of future design and implementation, as well as contractibility.

Although this approach addresses a significant subset of data requirements, it does not cover all of its aspects, typically the dynamic ones. Therefore, the RAINBOW approach does not replace more traditional task and information analysis approaches, but rather complements them. For instance, the form-based graphical representation of the underlying data schema can be used during interviews to stimulate the discussion.

As for the generated prototype, it can be used during the task analysis to capture real-time use cases and define the expected behaviour of the system. In addition, analysing how the tasks are performed using the prototype in comparison to the legacy information system (if any), can help to support the Reverse engineering of existing artefacts and even induce more general considerations on the definition of the target information system.

5. VALIDATION

To experiment and evaluate the approach, a validation protocol has been defined in order to notably assess its ability to help end-users and analysts to communicate static data requirements to each other, and the quality of the conceptual schemas it produces. The validation protocol relies on the Participant-Observer principles to monitor the use of the RAINBOW approach and toolkit, and the Brainstorming/Focus group principles to analyse the resulting conceptual schemas, as defined in (Singer et al, 2008). This protocol was used for a first series of experimentations where pairs of end-users and analysts were asked to jointly define the conceptual schema of a future information system using the RAINBOW approach and tool kit.

For each project, the first task consisted in preparing the experimentation by defining the subject based on real-life concerns of the end-users, then training the participants to understand the method and use the tools. Secondly, the end-users and analysts were asked to apply the approach on their project and focus on the five first steps, while observers took notes. The third step consisted in analysing the observations on the efficiency of the approach. Finally, the quality of the produced schemas was debated, taking in account schemas that were designed by the analysts without seeing the output of the approach.

The analysis of these experimentations, confirmed the validation canvas to be valid and relevant, as it notably highlighted that the RAINBOW approach and tool support did help end-users and analysts to communicate static data requirements to each other, while generating a positive response from the participants. Though all the requirements could not be expressed through the toolkit, the latter did serve as a basis for discussion and modifications. On the other hand, the quality of the conceptual schemas produced using the RAINBOW approach were relevant, understandable and comparable to the schemas that an analyst could produce by himself.

These early results are therefore encouraging, though special care should be given to improve critical aspects such as the assignment of responsibilities, the drawing behaviours, the customisation of the tools and the relevance of the elements they highlight. This preliminary validation process also stressed several sensible and interesting phenomena, such as the emergence of different design styles during the drawing phase, for instance regarding the grouping of elements in containers. Besides, given the intrinsic difficulty of evaluating methodologies for the development of large systems, the RAINBOW approach deserves to be tested through multiple settings over time and compared to other databases methodologies in order to study its true effects and monitor the previously mentioned phenomena.

6. CONCLUSION

In this paper, we presented the tool-supported RAINBOW approach, designed to interactively involve end-users in the database conceptual analysis process of information systems engineering and facilitate the communication between analysts and end-users. This comprehensive approach is based on the reverse engineering of user-drawn form-based interfaces to perform an interactive database conceptual analysis, and proposes to elicit and validate static database requirements based on end-users involvement through interactive prototyping, as well as the integration and adaptation of techniques coming from various fields of study.

This original and realistic contribution, aiming to elicit static database requirements and promote user involvement in the context of Information Systems Engineering, addresses the necessity to actively involve end-users of a future IT system during its specification and development, as notably advocated by the proponents of Participatory Design. Beyond the fact that end-users know "how business is done" in the environment for which an information system is being developed, such practices can help to avoid resentment and resistance towards a new information system infrastructure, as well as to stimulate productivity. Besides, the RAINBOW approach overcomes the main concerns raised by similar researches, while being interoperable with other approaches and extensible for further analysis and elicitation processes.

In the approach, the expressiveness of form-based interfaces and prototypes, combined with the specialisation and integration of standard technique to help acquire and validate specifications from existing artefacts, enable to use form-based interfaces as a two-way communication channel to communicate static data requirements between end-users and analysts. The current experimentations comfort us in believing that this approach is viable and worthy. By pursuing these experiments with the collaboration of end-users over time, we will be able to enrich and improve the approach, its processes and its tool support based on the feedback gathered and the analysis of the observed phenomena.

REFERENCES

Batini, C., et al, 1984. A methodology for conceptual design of office databases. *Information Systems*, Vol 9(3/4), pp 251-263.

Batini, C., et al, 1991. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.

Chi, Y., et al, 2005. Frequent subtree mining - an overview. *Fundamenta Informatica*, Vol 66(1-2), pp 161-198.

Chikofsky, E.J., and Cross, J.H., 1990. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, Vol 7(1), pp 13-17.

Choobineh, J., et al, 1992. A form-based approach for database analysis and design. *Communications of the ACM*, Vol. 35 (2), pp 108-120.

Cohen, W.W., et al, 2003. A Comparison of String Distance Metrics for Name-Matching Tasks. *Proc. of IJCAI-03 / IIWeb'03*, Acapulco, Mexico, pp 73-78.

Connell, J., and Shafer, L.I, 1995. *Object-oriented rapid prototyping*. Yourdon Press, Upper Saddle River, NJ, USA.

Davis, A.M., 1992. Operational prototyping: A new development approach. *IEEE Software*, Vol 9(5), pp 70–78.

Elmasri, R., and Navathe, S., 2006. *Fundamentals of Database Systems*. Addison Wesley.

Fellbaum, C., 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

Hainaut, J.L., 1989. A generic entity-relationship model. *Proc. of the IFIP WG 8.1 Conference on Information System Concepts: an in-depth analysis*, North-Holland, pages 109-138.

Hainaut, J.L., 2002. *Introduction to database reverse engineering*. LIBD Publishing, FUNDP, Namur, Belgium. Available from: http: //www.info.fundp.ac.be/~dbm/publication/2002/DBRE-2002.pdf

Hainaut, J.L., et al, 2008. Migration of Legacy Information Systems. *Software Evolution*, Springer, pp

107-138.

Hall, P.A.V., 1992. *Software Reuse and Reverse Engineering in Practice*. Chapman & Hall, Ltd.

Hersh, W., et al, 2000. Assessing thesaurus-based query expansion using the UMLS Metathesaurus. *Proc. of the AMIA Symposium*, pp 344-348.

Hick, J.M., and Hainaut, J.L., 2006. Database application evolution: A transformational approach. *Data Knowledge Engineering*, Vol 59(3), pp 534-558.

Jiménez, A., et al, 2008. Mining induced and embedded subtrees in ordered, unordered, and partially-ordered trees. *Proc. of the 17th International Symposium on Foundations of Intelligent Systems (ISMIS'08)*, pp 111–120

Kösters, G., et al, 1996. Combined analysis of user interface and domain requirements. *Proc. of the 2nd International Conference on Requirements Engineering*, IEEE Computer Society, pp 199–207.

Lopes, S., et al, 2000. Efficient discovery of functional dependencies and armstrong relations. *Proc. of Advances in Database Technology - EDBT 2000*, Konstanz, Germany, pp 350-364.

Nuseibeh, B., and Easterbrook, S.: Requirements engineering: a roadmap. *Proc. of the Conference on the Future of Software Engineering*, ACM Press, pp 35–46.

Pomberger, G., et al, 1991. Prototyping-oriented software development - concepts and tools. *Structured Programming*, Vol 12(1), pp 43-60.

Ramdoyal, R., 2010. *Reverse Engineering User-Drawn Form-Based Interfaces for Interactive Database Conceptual Analysis*. Phd Thesis, Presses Universitaires de Namur, University of Namur, Namur, Belgium, 2010. Electronic version available from: http: //info.fundp.ac.be/libd/rainbow

Ramdoyal, R., et al, 2010. Reverse Engineering User Interfaces for Interactive Database Conceptual Analysis. *Proc. of CAiSE 2010*, Hammamet, Tunisia, pp. 332-347.

Rode, J., et al, 2005. As easy as "click": End-user web engineering. *Proc. of the 5th International Conference on Web Engineering (ICWE 2005)*, Sydney, Australia, Vol 3579 of LNCS, Springer, pages 478-488.

Rollison, S.R., and Roberts, S.A., 1998. Formalizing the informational content of database user interfaces. *Proc. of the 17th International Conference on Conceptual Modeling (ER'98)*, pp 65-77.

Schewe, K.D., and Thalheim, B., 2005. Conceptual modelling of web information systems. *Data Knowl.edge Engineering*, Vol 54(2), pp 147-188.

Schuler, D., and Namioka, A. (editors), 1993. *Participatory Design: Principles and Practices*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA.

Shoval, P., and Shiran, S, 1997. Entity-relationship and object-oriented data modelling - An experimental comparison of design quality. *Data Knowledge Engineering*, Vol 21(3), pp 297–315.

Singer, J., et al, 2008. Software engineering data collection for field studies. *Guide to Advanced Empirical Software Engineering*, Springer, pp 9-34.

Spaccapietra, S., et al, 1992. Model independent assertions for integration of heterogeneous schemas. *VLDB Journal*, Vol 1(1) pp 81-126.

Terwilliger, J.F., et al, 2006. The user interface is the conceptual model. *Proc. of 25th International Conference on Conceptual Modeling (ER'06)*. Vol 4215 of LNCS, Springer, pp.424–436.

Vilz, J., et al, 2006. Data Conceptualisation for Web-Based Data-Centred Application Design. *Proc. of CAiSE 2006*, Luxembourg, p205-219.

Winkler, W.E., 1990. String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. *Proc. of the Section on Survey Research Methods*, pp 472–477.

Yang, F., 2008. WYSIWYG development of data driven web applications. *Proc. of the VLDB Endowment*, Vol 1(1), pp 163–175.