ABSTRACT
In this paper we explore the chances of designing components of information systems by making the most of explicit and implicit information gathered from human-computer interfaces. In this process we assume that end-users, with little training, are able to sketch an interface prototype for the future application, from which we can derive a conceptual model of the application domain in a semi-automatic way. In particular, information contained in electronic forms can be extracted and conceptualized through reverse engineering and integration techniques in order to significantly help to design the conceptual schema of the future database.

KEY WORDS
Database Design, Graphical User Interfaces, Software Requirements, End-user, CASE Tool.

1 Introduction
Prototyping is one of the most effective elicitation techniques. Actually, this method is particularly useful when the purpose of the future application is not clear enough, and when users’ opinion is required in the early stages of the design process [1]. Working out a prototype in the requirement analysis phase is obviously an expensive task: it includes the development of a limited but operational application, whose components cannot usually be reused in the implementation of the final system.

In recent years, the understanding of user interfaces has improved in organizations thanks to the increasing use of information technologies and a better training level in this field. On this basis, we can assume that users are now quite familiar with GUIs and that many of them should be able to draw an interface prototype for the future application. Such a prototype may bring essential information especially on the static objects of the application domain and on users tasks.

Therefore it is possible to make end-users take a more active part in the expression of their needs. They are asked to build an interface for the future application in terms of command screens and information exchanges. Our goal is to derive task and functionality models from these components as well as an underlying information model.

This paper mainly focuses on the problem of extracting static objects specifications from a prototype interface, that is, designing the conceptual schema of the database (or at least contributing to its design) through user interface analysis. Due to the limited scope of this paper, the three following principles we will be taken as hypotheses:

1. from the user’s perspective, forms, and especially electronic ones, are a more natural and intuitive way of expressing information requirements than usual conceptual formalisms;
2. a form contains data structures that can be seen as a particular view of the conceptual schema of the database;
3. a form is a physical implementation of a part of the conceptual schema, so that database reverse engineering techniques can be used to recover that part of the schema, as illustrated by Astrova and Stantic in their approach focused on ontologies [2].

The proposed approach consists in letting users draw an interface that most suit their needs for the application-to-be, then in analyzing the resulting dialog boxes and forms in order to extract their underlying information structures. From the latter, a conceptual schema is progressively built and validated. Although this approach does not entirely replace more traditional methods dedicated to task and information analysis, nevertheless, it allows to gather more efficiently and more reliably a substantial part of initial specifications.

The method described in this article addresses the analysis and development of databases. In order to reduce costs and improve the quality of the requirements analysis phase, it provides designers and end-users with tools that make conceptual schema design easier.

The phase that deals with extraction and modelling of the behavioral aspects of the target application through interface analysis will not be discussed in this paper.

The paper is divided in two main parts: the first one describes the theoretical aspects of our approach and the second one illustrates it with a small case study.
This method is being developed within the context of the ReQuest project, whose objective is to create intelligent tools dedicated to the development of web-based applications.

2 The approach

Figure 1 illustrates the tasks of our approach using the ConcurTaskTree notation [3]. Other elicitation techniques, such as natural language analysis and interviews have been ignored for simplicity reasons.

The main task (Specification), which aims at designing the conceptual schema of the application domain, is divided in sub-tasks. Requirements elicitation comes first. It includes two main steps: interface drawing by end-users and conceptualization.

According to his/her skill, several WYSIWYG tools dedicated to interface drawing are made available to the user. These tools range from a freehand sketching tool on a tablet PC, which provides fast and easy, but low-fidelity results, to comprehensive interface builders, which can produce a high-fidelity prototype that will remain almost unchanged in the operational system. After a short training, the user is able to build the interface (screens, dialog boxes, forms, web pages) he thinks the most appropriate according to the requirements of the future application.

Two extreme scenarios are considered for this task:

- the task is performed by the user alone, provided he masters the drawing tools and the elementary rules of interface design;
- the task is performed jointly by the user and an experienced designer who translates users’ requirements into an interface.

Other intermediate patterns can be defined according to the skill of end-users in interface design and the availability of experienced designers.

According to the semiotic approach of user interface design [4], a user interface is considered to be a complex message exchange between the system and the user. In order to identify the concepts that the future application will use, we have to identify the semantics of these messages; this is what the “conceptualization” task aims at. Such messages are made of signs that take the shape of widgets in graphical user interfaces. Though some of them are highly expressive and therefore semantically rich, the user will often be asked to add interpretation annotations that will contribute to a more comprehensive conceptualization.

The concepts conveyed by screens and forms are abstract entities that belong to the mental model of the user. These entities are represented in a standard model [4] based on the Entity Relationship (ERA) model. Raw diagrams representing data structures are built during the “importation of user interfaces” sub-task. They give a neutral representation of underlying data structures but are not interpreted yet.

When carrying out the “Interpretation” sub-task, the analyst elicits the domain concepts using raw diagrams derived from user-drawn interfaces. He also identifies and solves gaps and incoherences between these concepts. Afterwards, assisted by the user, he will correct and refine user interfaces. This process is iterative by nature.

After total validation from the user and the analyst, the diagrams produced by the interpretation of interfaces are integrated to form a global schema of the application domain.

At this stage, the user will often be asked to help to perfect the specifications and remove remaining ambiguities.

2.1 Drawing interfaces

In the proposed approach, descriptions of interfaces are a major source of information for conceptualization. These descriptions are used in two ways. On the one hand, they are analyzed in order to extract the conceptual schema of the future database, and on the other hand they are used to generate the actual user interface for the application according to a target technology. The description format has to comply with a strict syntax that is independent of any operational tool dedicated to user interface management. This property makes it possible to use code generators for many languages and graphical user interface libraries.

Within the context of the Request project, the USIXML language [5] meets these characteristics, while the GrafiXML WYSIWYG tool [6] makes it possible to draw user interfaces and uses this language for their specifications.

The user is asked to use simple and intuitive coherence rules while designing interfaces. Some important rules suggest:

- to assign an expressive label to each control and to place it at one from definite positions;
• to group all fields that relate to a same concept into a common frame;
• to assign the same label to controls that refer to the same concept (as few synonyms as possible);
• to assign different labels to controls that refer to different concepts (as few homonyms as possible);

These rules are not always applied because of ergonomic reasons (the user may not wish to assign an explicit label to a control that doesn’t require one) or in case of ambiguities (a customer and a supplier both have an address, but these addresses do not represent the same concept). In such cases, the user can annotate the control and assign to it an hidden label, possibly with the help of the analyst. We therefore dissociate the visible interface label from the invisible semantic label.

2.2 Importing interfaces

Several approaches have explored the existing link between the user interfaces and the conceptual schema of an application domain both in forward engineering [7][8][9] and in reverse-engineering [10]. The "importation of user interfaces" task is one of them. It takes into consideration the evolution and ergonomics of graphical user interfaces.

"It is necessary to understand the semantics of each control in its different uses in order to be able to transform user interfaces into a semantic data model" [8]. For this purpose, we use the ergonomic rules of [11].

Some of these rules allow to choose the right control according to the context. For instance, one of them recommends to use a group box containing checkboxes to represent a choice among four to seven known alphanumeric values.

A set of rules determines patterns for using controls. We have associated a conceptual representation to each pattern derived from common ergonomic rules. The previous example is represented by a multivalued attribute that takes the name. Its domain of value is made up of the checkboxes labels. A pattern matching mechanism that uses these rules automatically transforms XML interface descriptions into ERA uninterpreted raw diagrams.

Some patterns require the use of additional annotations when the user interface structure is not sufficient to allow semantic extraction. Metaphoric display is one of these patterns. It uses a real world representation to make the user task easier. Annotations are required to help the interface importer to understand these metaphors.

For example: "business, accountancy and massive data acquisition applications can use the filling-the-blank metaphor" [11]. In this metaphor, the concept of label has no meaning. For the user, the semantics comes from the whole sentence containing the blanks. The interfaces importation tools require a hidden label for each blank. In Figure 2, these labels could be "Beginning of license validity period" and "End of license validity period".

![Figure 2. Example of the "Filling-the-blank" metaphor](image)

2.3 Interpretation

The interpretation task consists in identifying the relevant concepts of the application domain in the uninterpreted schema.

We consider a concept to be potentially relevant if it shows some degree of structural complexity: a concept can for instance be represented by an entity type (derived from an interface screen) or a compound attribute (group box or table). It is important to distinguish the structures forming a local complex property of the object described by the window from the structures referring to relevant concepts that are external to this object. As in the conceptualization stage of the reverse engineering method described in [12], this distinction highly depends on the semantics and on the application domain. Hence, the user's assistance is often necessary.

For example, the concepts "customer address" and "customer orders" can both be represented by multivalued compound attributes. After further analysis, it may appear that,

• the "address" complex attribute is interpreted as a local customer property,
• the "orders" complex attribute is interpreted as an external concept that must appear in other interfaces.

Interpretation then consists in representing all identified concepts by entity types according to the rules established in section 2.1. A compound attribute that appears as a compound attribute or an entity type in another interface is extracted as an autonomous entity type according to the transformational approach [13]. Entity types take the name of their hidden label, if any, or that of their visible label otherwise.

Gaps and schema incoherences are highlighted in this automatic phase. Then, the analyst completes and corrects the schema with the user’s assistance in order to deal with two aspects. The first one is the semantics of the link between the new concept and the interface from which it is extracted. This link is documented in the schema with a name given to the corresponding relationship type. It must be stated that the characteristics of relationship types are not complete at this stage, since the cardinality of one role is still undefined. It will however be set during the integration phase. The second aspect is the relative identification of the concepts with regard to the interface (for example, can a same product number appear several times in an order form?).

These modifications made to interpreted schemas are reflected in the interfaces through the addition of semantic
annotations, and a new version of the schemas is generated from these supplemented interfaces. After several iterations of the Requirements elicitation task (Figure 1), the annotated interfaces form the complete and non ambiguous documentation of the concepts that the future application will use. A database could therefore automatically be generated from this documentation.

2.4 Integration of interpreted schemas

The user-validated schemas are integrated to form a conceptual schema that represents the requirements specification expressed by the user through interfaces. This specification should be formal, complete, minimal, unambiguous, abstract and readable. The domain concepts and their internal links have been univocally represented by similarly named objects throughout the extracted and interpreted schemas. Concepts are represented by entity types while links between them are represented by relationship types. The integration of these objects can be automated, to a large extent, given the interface-derived preparation of partial schemas.

During this phase, relationship types characteristics are refined. For instance, let us assume that an order can be associated with several invoices in a given interpreted schema, and that an invoice corresponds to one and only one order in another interpreted schema. The two semantically similar relationship types will be integrated into a single relationship type, whose role cardinalities are clearly defined: an order is associated with several invoices and an invoice corresponds to one and only one order. Consequently we have a one-to-many relationship type from order to invoice in the integrated schema.

After this automatic phase, the analyst works jointly with the user in order to fill any remaining gaps left undetected or unresolved in previous phases. Usually, this job mainly handles issues related to absolute identifiers and cardinality definitions. These modifications will also be reflected in the interfaces thanks to annotations.

Our integration rules follow those that have been proposed in the database community [14][15]. However, they also have to cope with incomplete specifications, so that conflict-solving rules have been adapted.

3 Case Study

This short illustration deals with the organization of training courses. We will show how we put into practice the first three stages of the proposed approach.

The three first steps are represented by Figure 3. The result of the last step is shown in Figure 4.

3.1 Drawing interfaces

The user draws the prototype of two windows that are considered necessary and sufficient to deal with the part of the future system that records and manage courses. The "Session" window gathers information on a particular course session and we know that each course can be given several times. A session is split into modules and each one of them has its own window. Each "Module" window allows the user to collect specific information on these modules, such as the teachers' names.

3.2 Importing interfaces

The information content of each window is abstracted according to the ERA model by applying pattern matching rules. Technically, the windows drawn by the user with the help of the GrafiXML tool are exported to a USIXML document. This document is then imported in the DB-MAIN CASE tool [16] and interpreted to produce an ERA schema. The "Session" window is represented by the "Session" entity type. Group boxes ("Fare", "Course", "Module") are represented by compound attributes. A group box containing several fields is represented by a single-valued attribute, while a group box containing one and only one table becomes a multivalued attribute, whose sub-attributes correspond to the table columns. Edit boxes ("Number", "Date", "Number of days", "Name", "Language", "Date") are represented by atomic single-valued attributes. Each one of them takes the name of the corresponding identifying label.

3.3 Interpretation

Relevant concepts (as defined in section 2.3) are then transformed into entity types. This transformation is automatic since the documentation obtained during the Importation of interfaces task is complete and non ambiguous. In this case the relevant concepts are: the "Module" compound attribute in the "Session" entity type, the "Course" compound attributes in the "Session" and "Module" entity types, and the "Session" compound attribute in the "Module" entity type. They are transformed into entity types using DBMAIN transformation primitives.

After this transformation, the analyst has to detect missing information such as the semantics of the new relationship types and the cardinalities of the new roles. For instance, the relationship type linking "Session" and "Module" and the cardinality of the role played by "Module" must be defined. According to a discussion with the end-user, the interface is enriched with annotations that add the acquired information, and the sub-schemas are finally validated. In particular, the relationship type between "Session" and "Module" is named "composition". The unknown cardinalities, printed in bold type in Figure 3, remain unchanged because some of them will automatically be resolved during the integration phase.
3.4 Integration of interpreted schemas

The integration process follows the pre-integration approach, during which all the conflicts that may arise during the final merging of schemas are resolved.

Since, on the one hand, sub-schemas have been prepared in order to name all concepts univocally (no homonymy, no synonymy) and, on the other hand, each concept has been semantically validated, the final phase is automatic. Hence, we obtain a provisional conceptual schema of the application domain. The missing information that has not been resolved during previous tasks is shown in bold type in the results of the integration step (Figure 4). In this case, the analyst is assisted by the user to validate the proposed cardinality \([0-N]\) that represents the number of sessions for a course. Figure 4 shows the final conceptual schema.

4 Method extension

We can achieve two more tasks from the complete screen specification and the final conceptual schema, namely the design of the database and some procedural components of the future application.

4.1 Database Design

The database is designed from the conceptual schema using a traditional approach based on transformational techniques. Traceability is ensured by the transformation history that is recorded and maintained during the whole process, thus allowing instant association between interface components and database objects. This history allows automatic SQL queries generation to fill user interfaces with existing data.
4.2 Conception of procedural components

Natural business objects can also be derived by the analysis of similarities between interface interpreted sub-schemas. We then obtain object schemas from which we can automatically generate object classes code, including constructors, accessor and modifier methods, as well as ready-to-implement frameworks of processing methods. In order to implement the latter, business rules must be taken into account (this part of the project has not been explored yet.). The connection between interface components and database objects will be ensured by the obtained business objects.

5 Conclusion

The results of our approach allowed to define and extract all necessary information contained in user defined interfaces in order to create the conceptual schema of the application domain as well as important software components. The application domain model is not sufficient however to elaborate a complete information system. Other aspects must be taken into account as in standard analysis methods.

From a practical point of view, some tools have been developed in the DB-MAIN case tool to evaluate our approach: an interface importation tool, a schema interpreter, a schema integrator and a queries generator.

The extension of our method consists in analyzing information included in human-computer interfaces and useful for designing enterprises organization models, data models, functional and non-functional models. The available backtracking in the method enables to highlight existing links between models, and for instance to generate database queries based on the functional and the data models.

References


