Data Conceptualisation for Web-Based Data-Centred Application Design

Julien Vilz, Anne-France Brogneaux, Ravi Ramdoyal, Vincent Englebert, and Jean-Luc Hainaut

Laboratory of Database Application Engineering - University of Namur,
Rue Grandgagnage 21 - B-5000 Namur, Belgium
{jvi, afb, rra, ven, jlh}@info.fundp.ac.be

Abstract. The paper describes the conceptualisation process in the ReQuest approach, a wide-spectrum methodology for web-based information systems analysis and development. This methodology includes a strong involvement of end users in the requirement elicitation process by building prototype user interface fragments of the future application. The paper focuses on the analysis step of these fragments that yields a draft conceptual schema of the application domain. The analysis includes a tree-based representation of the fragments, the detection of shared subtrees through mining techniques, their normalisation and the derivation of the conceptual schema. A short description of a supporting tool is given.

1 Introduction

Despite a large offer of open source and proprietary IDEs for developing web-based applications, the problems of designing and developing quality applications at reasonable cost still are open issues, as testified by the poor quality and the high cost of many e-commerce applications.

The ReQuest framework is a wide spectrum tool-supported methodology for web-based data-centred applications analysis, development and maintenance intended to address these problems. It relies on five principles, namely, (1) intensive user involvement in the requirement collection and analysis phases [1], (2) the induction of the system behaviour from users scenarios [2], (3) the induction of the conceptual data structures from user interfaces [3], (4) the wrapper-based integration of existing data and services [4], and (5) automated validation and generation of major components of the target application. This research is part of ReQuest, a project funded by the Region Wallonne.

This paper concentrates on some important aspects of the database stream of the methodology, and more particularly on principles (1) and (3), that state that a large part of the conceptual data structures can be inferred from the user interface. Whereas existing approaches for modelling web information systems [5] usually produce presentation and navigational models from an existing conceptual schema built using the most traditional approach [6], this elaboration of the conceptual schema is a key part of the ReQuest approach.

E. Dubois and K. Pohl (Eds.): CAiSE 2006, LNCS 4001, pp. 205 – 219, 2006.

According to principle 1, representative users are invited to sketch, through intuitive drawing tools [7], the user interface (comprising windows, dialog boxes, electronic forms and task description) to the intended system that best suits his/her needs. Though this *low-fidelity* description still has to be validated and modified with the help of design experts, it provides a rich description from which the most important components of the database schema can be extracted.

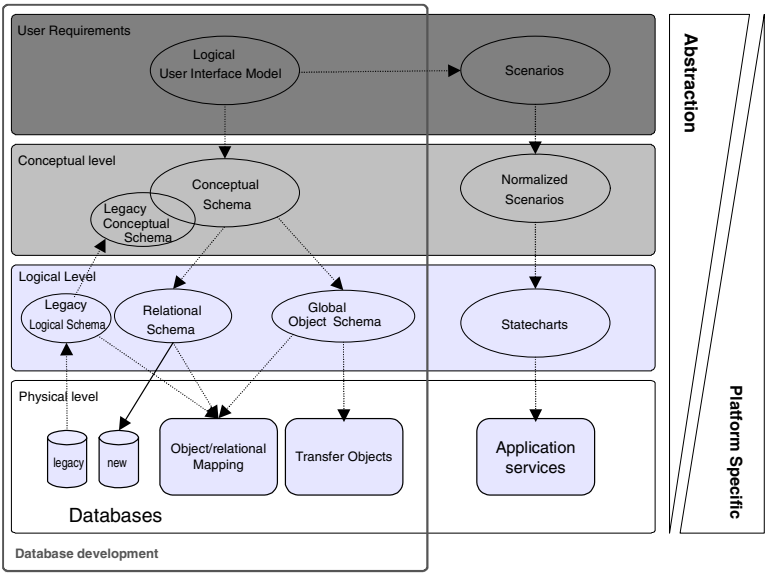

Fig. 1. The ReQuest framework as an MDE-compliant approach. The left area of the schema represents the products of the database development stream. Arrows express derivation processes between design products and artefacts.

Fig. 1 shows the part of the ReQuest framework that copes with database building. The design and development of the application services (behavioural aspects) are only suggested at the right part of the schema while the user interface development (interaction aspects) is not shown.

In the ReQuest approach, prototype interfaces sketched by users are translated into a *Logical user interface model* that, among others, expresses in an abstract way their underlying data structures. This logical interface model is then analysed in order to identify redundant data aggregates that the user naturally exchanges with the system, to reduce these potentially conflicting structures and to produce the conceptual schema. The *Scenario models* (right) and the kernel of the *Conceptual schema* of the database (left) are derived from the Logical interface model. From the latter, we derive the relational schema of the new database as well as the *Object schema* through which the application services will access the legacy and new data.

Integrating existing services is sketched for data only. The logical schema and the conceptual schemas of the *legacy database* are extracted through reverse engineering

techniques [8]. The conceptual schema is compared with the global conceptual schema and the subset of the latter that is not covered by the legacy data is translated into the schema of the *new database*. At the physical level, legacy data access is ensured through a wrapper-based interface [4].

These derivation processes heavily rely on the transformational paradigm, which guarantees the propagation of specifications throughout the various levels of abstraction, from user requirements to the application code [9].

Analysing the information underlying corporate forms has long been a part of database design methodologies [6]. Extracting data structures from user interfaces has been proposed in [10] and eliciting knowledge from forms is described in [11]. The former proposal describes structural derivation rules, induction rules from sample data and an expert system that extracts a tentative conceptual schema from a set of forms. The latter proposals is based on a tree representation of forms that allows tree manipulation algorithms to be applied. In the database reverse engineering realm, [12] develops the FORE method to capture the knowledge buried in forms and to derive from it a conceptual object-oriented model.

The specific aspects of Web applications have lead to several specification methodologies that generally are extensions of standard approaches. Entity-relation models have been used and adapted in [13] and in [14] for instance, while [15] proposes an extension of the ORM model. The user-driven dynamic aspects of these systems has lead to enrich these models with behavioural specifications [16]. WebML [17] integrates most pertinent aspects of web applications and makes them available through models, modelling languages, processes and graphical tools. However, inference mechanisms such as those developed have not been developed so far.

Finally, the importance of user involvement in Web application design has been emphasised by, e.g., [18] and [19].

This paper is organised as follows. Section 2 describes the aspects of user interfaces that are relevant to database design and their mapping to the Logical interface model. Section 3 describes the conceptual extraction process, while Section 4 concludes the discussion.

2 User Interfaces as User Requirements Expression

The most abstract level the ReQuest approach is the *User requirements level*. From users viewpoint, electronic forms and dialog boxes appear to be the most natural form of system description. Their use and structure are familiar to end users [11] and the transition to a semantic model has been shown to be tractable [10]. Indeed:

- forms are more natural and intuitive than usual conceptual formalisms to express information requirements;
- the data structures contained in each form can be seen as a user view of the (future) conceptual schema of a database;
- finally, since a form is a kind of physical implementation of a part of the conceptual schema, database reverse engineering techniques can be used to recover that part of the schema [8].

The information on the components of user interfaces, such as the type of the component, explicit labels, semantic names, value type, value size, sample data, default value, and dynamic links, is exploited in two ways.

First, it is analysed to extract *user objects* that are natural data component aggregates that appear in forms. The latter are then transformed into the conceptual schema of the future database. Discovering the semantic of the user objects is similar to a reverse engineering process applied to the underlying data structures of the forms.

Secondly, it is used to specify the task model and to develop and generate the actual user interface of the target application. These aspects are ignored in this paper.

The input of the conceptualisation process is the Logical user interface model, which is extracted from the physical code of the prototype interface. In the ReQuest approach, it consists of a tree representation of restricted web forms.

2.1 User Interface Drawing

Strong involvement of end users in the requirements elicitation process is one of the major objectives of the ReQuest approach: this aspect is materialised by allowing the end users to design a prototype user interface of the future application. To this aim, selected users must have a good knowledge of the application domain and must be familiar with form-based application interface. They are briefly trained in the use of the interface drawing tool and are taught simple guidelines in user interface design. These guidelines aim to ease the automatic analysis of user interfaces and to reuse them in the final application . The guidelines target web-oriented systems. For instance they encourage users to avoid tables for layout purpose in web pages.

The user builds the interface either alone, or with the help of an experienced designer. Empirical studies have shown us that motivated users can quickly design quite valuable prototype interface [20].

Web pages combine specific content and reusable template material [21]. Therefore, the end user designs and annotates the different fragments of the interface according to the context (navigational information, domain application information as well as states of performed tasks), then assembles them to obtain the different complete web pages. Later on, the user can be asked to provide sample data as part of the specification process, which could notably lead to highlight functional dependencies [22].

2.2 Logical User Interface Model

The logical model of an interface comprises controls as well as input and output information for a task that will be performed by the target application. For each interface fragment drawn by the user, the corresponding tree-structured logical user interface model is created, following the translation rules of Table 1.

The data part of the logical interface model is a forest. Leaf nodes are labelled with the user name of the corresponding logical input or output interface components. The (hopefully) semantic name is either extracted from the form labels or explicitly provided by the user when drawing the interfaces. It identifies the concept handled by the component (for instance a *name* or an *address*).

A non-leaf node represents a logical container such as a table and group box. Its subtree gathers the data handled by the graphical components enclosed by this container.

The node representing the user interface fragment (the top container) is the root of the tree. Fig. 2 illustrates the modelling of two user interface fragments. Controls such as buttons add are ignored in this discussion.

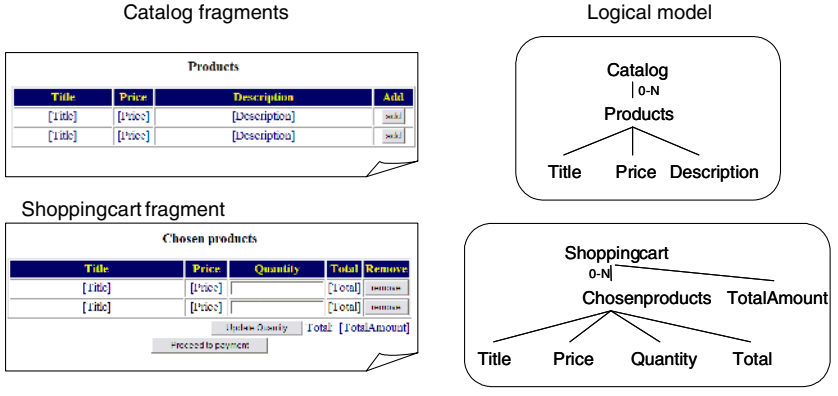

Fig. 2. The logical models of two user interface fragments. The data structures underlying each of them is expressed as a composition tree.

Tools have been developed to extract this model from user interface components written in XUL and XHTML. We have also extended these languages with custom properties to support the necessary annotations described in Section 2.1.

Table 1. Common widgets and their tree translation

Widget	Tree equivalent	Properties
Group box	Node whose children represent the elements contained in the group box.	name
Text input	Leaf node.	name, maximum length, value type, mandatory
Radio button set	Leaf which aggregates the radio buttons of the set.	name, value domain
Check box set	Leaf which aggregates the check boxes of the set.	name, value domain
Selection list	Leaf node.	name, value domain, multiple, mandatory
Table	Node whose children represent the elements contained in the table. The table is dynamic if its content is generated on demand	name, dynamic; if relevant, maximum length, value type, mandatory of the cells

2.3 Formal Definition of the Logical User Interface Model

Each node of a tree in a model is *labelled* with the properties extracted from the widgets, e.g., the semantic name, the size of the input data or the value domain of a selection list. Hence, the formal definition of this model is an extension of the definition of rooted labelled tree as defined in [23].

To cope with the properties of the data structure extracted from the user interface, the labelling function needs to be refined in order to distinguish the two following properties: the semantic name and the value domain of the data.

Therefore, we define the set *Att* of pairs N x D where N is a set of names and D a set of value domains.

- The function A: V→ (N, D) replaces function L.
- The functions NameOf : *Att* → N and DomainOf : *Att* → D allow each part of the definition of an attribute to be accessed.

A new labelling function for the edges is also needed to represent repetitive data structures such as tables or optional data structures. The function C: E → *Card* attaches cardinalities to edges. *Card* is a set representing the minimum and maximum number of child node instances allowed for each parent instance (Fig. 2). Typically,

- *Card* = {0-1, 1-1, 0-N}

The definition of a tree in our logical user interface model hence becomes: T (V, E, Att, Card, A, C).

3 From the Logical User Interface Model to the Conceptual Schema

3.1 Searching for Domain Concepts

A domain concept can appear in several interface fragments. Since a user view describes a user object, we conclude that a user objects is made up of aggregates of domain objects that can appear in other user objects. Hence the idea to identify the subtrees that appear in more than one interface fragment of the logical interface model and to derive from them tentative domain concepts. We call them *shared subtrees*[1].

The basic procedure we will describe consists in (1) identifying the shared subtrees, (2) for each of them, removing all its instances from the source fragment, (3) representing the domain concept subtree by a standalone tree and (4) linking the modified source fragments to this tree. Moreover, the composition relationships that hold among shared subtrees in an interface tree is interpreted as semantic relationships between corresponding domain concepts.

The use of shared subtree limits the search for data structures to redundant tree structures. So a non-connex redundant group of nodes will not be identified in this approach, which can be extended by the use of wildcards in tree labels during the search, as in XML queries [24].

Subtree Definition. There are several types of subtrees, but the most suitable for our purpose are the *induced subtrees.* An induced subtree keeps the parent/child relationship of its source tree. Chi et Al. [25] define the *induced subtree* as follow: "For a tree T with vertex set V and edge set E, we say that a tree T' with vertex set V' and edge set E' is an *induced subtree* of T iff, (1) V' ⊆ V, (2) E' ⊆ E, (3) the labelling of V' and E' is preserved in T'. […] Intuitively, an induced subtree T' of T can be obtained by repeatedly removing leaf nodes (or possibly the root node if it has only

[1] The standard name in the domain of tree mining is "frequent subtree".

one child) in T." Basically, an induced tree is a view on the conceptual schema to be discovered, that is, a subset of its objects.

Identifying Shared Subtrees. Comparing the subtrees that can be derived from the forest of a user interface can be done by *mining* the logical interface model for shared subtrees. Several algorithms have been developed to solve this complex problem [23]. We choose the algorithm *FreqT* [25] since it has a reasonable complexity[2], and the output of the algorithm identifies each occurrence of each shared subtree.

FreqT handles ordered induced subtrees. Since the order of sibling nodes is immaterial, we can, without any information loss, order them as required, for instance according to the lexicographic order on the set N used in the labelling function A.

The algorithm *FreqT* builds potential induced subtrees of the trees extracted from the interface model. These subtrees, also called pattern subtrees, are built using redundant labels. If it exists several occurrences of a pattern subtree in the input trees this pattern is used to create new patterns by adding nodes to it.

According to the definition of induced subtree, the labelling of the source tree must be preserved in T'. As we have redefined the labelling function in our logical user interface model, the equivalence between node labels must also be redefined.

Theoretically, two nodes v_i and v_k are declared identical if the set of input or output data instances in the widgets they represent are identical, or at least, if one set is a non empty part of the other one. However, since the logical interface model is extracted from *unpopulated* interfaces, we can only rely on structural and naming properties.

In the context of conceptualisation, we approximate $v_i = v_k$ with $n_i \cong n_k$ and $d_i \approx d_k$, as follows (n_p = NameOf(A(v_p)) and d_p = DomainOf(A(v_p))).

1. The *name similarity* relation "≅" defines a non strict equality between names. Since the same domain concept can have different names in several interfaces according to the context, the name equality relation can be refined, for instance to cope with synonyms and homonyms using dictionaries or ontologies. We currently use the *Jaro-Winkler* metrics [26] to compare node names. With this approach, "Chosen products" is similar to "Product" with a distance of 0,68 (two identical strings have a distance of 1). String distance metrics allow to highlight similarly strings according to different criteria. The user validate or invalidate the discovered similarity. As we have a complete control on input trees, a more sophisticated approach has not been felt to be useful so far.
2. The *domain similarity* relation "≈" is also defined as a non strict equality between domain values. It is verified whenever $d_i \subseteq d_k$ or $d_k \subseteq d_i$. This definition is driven by the fact that the logical user interface model does not always provide the precise domain values available for a specific widget. For instance, text fields can typically be used to input several data types such as names, numbers and dates. The user can specify the exact data type through annotations. Otherwise, the usual compatibility rules are used.

For instance, two user interface fragments include a field named "City", that, in the first fragment is defined by a predefined value list and in the second one as a 50

[2] *O(fmn)* with *f* the number of frequent trees, *m* the number of nodes in the largest tree and *n* the number of nodes.

character string. If all the predefined values are no longer than 50, then both domains are considered similar. We do not use the edge labelling function C in the induced subtrees research, but this information will be used later.

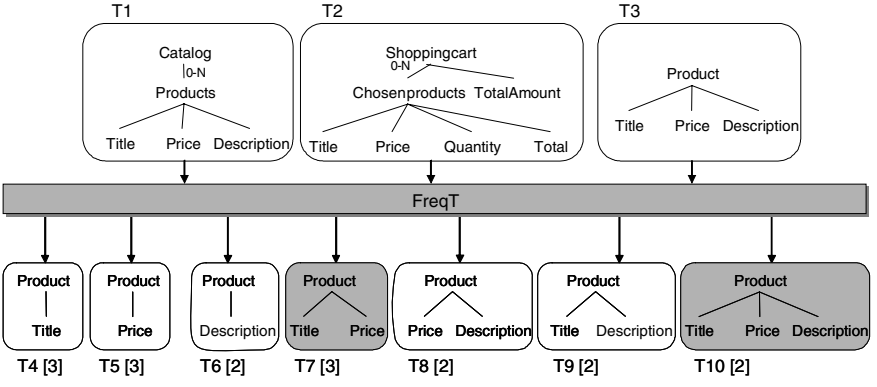

Fig. 3. *FreqT* extracts shared trees T4 to T10 from input trees T1 to T3. The number between brackets indicate in how many input trees the shared tree has been found.

Results of the Induced Subtrees Search. Algorithm *FreqT* produces a list of subtree patterns with the references of the instances that appear in the logical model. According to our goal, which is to isolate domain concepts in user interfaces, we will see that not all results are pertinent.

Fig. 3 shows the results of *FreqT* on the input trees T1, T2 and T3. For simplicity, we only keep the shared subtrees that have more than one node. We get seven subtrees T4 to T10, that all describe the potential domain concept "Product". We first observe that keeping only the *maximal induced subtrees*, i.e., those which are not an induced subtree of another result, is not necessarily appropriate. For instance the maximal induced subtree T10 is shared by two input trees only, while T7 is shared by all of them. We keep T7 and T10 and discard the other ones.

To keep only a set of results that will be useful for the transformation of trees into conceptual structures, we apply the following filter on *FreqT* results. Let

- T_i and T_k be shared subtrees resulting from application of *FreqT*.
- T_i be an induced subtree of T_k.
- I_i be the set of trees in which T_i appears (same for I_k and T_k).

T_i is retained if I_i and I_k are distinct tree sets, so that there is at least a tree in which either T_i or T_k appear.

Applying this filter yields subtrees T7 and T10. This result highlights links between potential domain concepts "Catalog", "Shopping cart" and "Product".

3.2 Representing Trees in the Entity-Relationship Model

Since most database designers are not familiar with abstract graphs, we express the data structures of the logical interface models in a wide-spectrum variant of the popular

Entity-relationship model, called the Generic ER model (GER), that encompasses logical and conceptual structures [9, 27]. In this section, we show how logical interface models and induced subtrees can be described in the GER.

In [27], the GER has been given a *non first normal form* (N1NF) interpretation. Trees as we defined them also are N1NF data structure. For instance,

Shopping Cart (Chosen products[0-N](Title, Price, Quantity, Total), TotalAmount)

is a N1NF relation schema expressing both the tree and the entity type of Fig. 4.

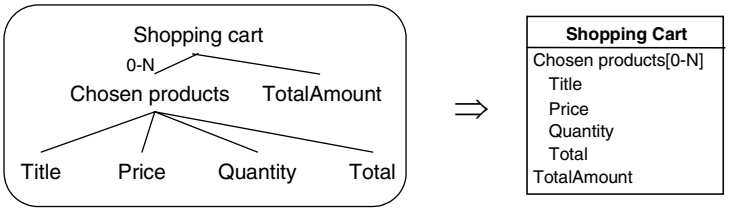

Fig. 4. Expressing a tree as an entity types with compound and/or multivalued attributes

Representing Shared Subtrees. The usual way to extract attributes from a relation is by *projection*. The projection of a N1NF relation seems the natural way to define a subtree from a source tree, provided this operator can still yield a N1NF relation. The standard projection produces *bottom-up subtrees*, as defined in [23]: "A bottom-up subtree T' of T is obtained by taking a vertex v of T together with all the descendants of v and their corresponding edges". For instance, in Fig. 3, we can extract subtree T10 from T1 written as N1FN relation using a projection[3]:

- T10(Products(Price, Title, Description))
- T1(Catalog (Products[0-N](Price, Title, Description))
- T10(Products(Price, Title, Description)) = T1[Products]

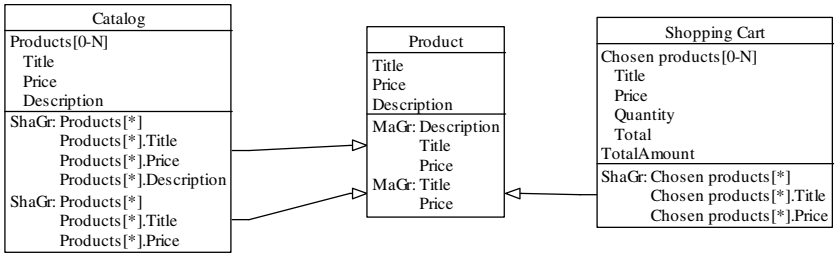

Fig. 5. Induced subtrees in entity types

[3] We only use an informal notation that is sufficient for the needs of the presentation. Notation R[I] specifies the projection of relation R on subset I of its attributes. More detail can be found in [15].

However, this operator cannot extract T7 from T1. So we define the *induced projection* to specify induced subtrees by indicating subtrees instead of mere node names. This projection is denoted by a N1FN relation representing the induced subtree to isolate and surrounded by "[ ]".

– T7(Product (Title, Price)) = T1 [Products(Title, Price)]

In the GER, the induced projection is represented by a group of attributes called *Shared Group* (ShaGr) in the third compartment of the representation of an entity type. The group gathers all the attributes involved in the projection[4]. Some induced projection can involve the entity type name itself like the projection that extracts T7 from T3 in the Fig. 3. We also define another type of group called *Master Group* (MaGr), which has the same meaning as the *Shared Group*, but indicates that the root of the induced subtree is the entity type itself.

According to our hypothesis, redundant structures such as "Product" in user interfaces probably represent domain concepts. An entity type named "Product" must therefore be created, containing a *master group*. A domain concept must be represented by one entity type, so that a shared subtree will be represented by one *master group* and one or more *shared groups*.

Fig. 5 represents trees T1, T2 and T3 of Fig. 3 by entity types Catalog, Shopping Cart and Product, and induced subtrees T7 and T10 by shared and master groups. An arrow is drawn from each shared group to its corresponding master group. Conceptually, this link is similar to a N1NF foreign key.

3.3 Logical Model Normalisation

The resulting schema must be processed in order to discard redundant specifications. We describe two techniques : embedded shared groups and redundant master groups.

Embedded Shared Groups. A shared group is embedded into another one when its components also are part of the latter. If they reference the same master group, then the embedded group can be discarded.

For instance Fig. 6 shows two links between "Catalog" and "Product", based on T7 and T10. Since the group that represents T7 is embedded into that of T10, we can discard it with no information loss.

Redundant Master Groups. During the transformation of trees and induced subtrees into entity types, several resulting entity types may have names such that $n_i \cong n_k$. We can assume that these entity types describe the same domain concept.

Fig. 6 extends Fig. 5 with a user interface fragment that allows the administrator to look at the "Shopping Cart" of the connected users. This fragment is called "Shopping cart (Administration)". If we assume that "Shopping cart (Administration)" ≅ "Shopping cart", there exists a shared induced subtree having "Shopping cart" as its root. That induced subtree is represented in both entity types by a *master group*.

[4] Symbol [*] means that the property holds for each instance of the parent instance.

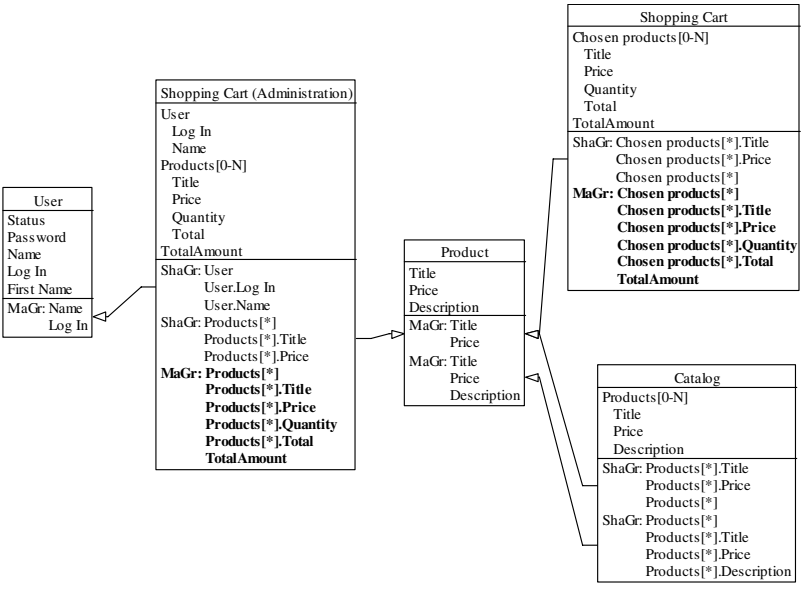

Fig. 6. An unnormalised schema that still includes redundant structures

As only one *master group* may exist for each shared induced subtree, we merge the entity types in which these groups have the *master* status[5]. Entity types integration keeps the *shared groups* of all merged entity types.

The integration consists in adding all the attributes which are not concerned with the redundant *Master Group* into the resulting entity type, as well as the groups involving those attributes. In the example Fig. 6, "User", "Login", "Name" will be added to the new entity type "Shopping Cart", as well as the group linked with the entity type "User".

3.4 Logical to Conceptual Transformation

The representation of trees and shared subtrees provides us with a logical Entity-relationship schema from which a pure conceptual schema must be extracted. This process is a variant of the conceptualisation phase of reverse engineering, through which one attempts to recover the conceptual origin of each technical construct. Transformational techniques have proved particularly powerful to carry out this process. The process is based on three main transformations.

1. Each shared group and its attributes are transformed into relationship type R with the entity type of the Master group. The transformation removes an attribute only if it does not appear in another, still unprocessed, s*hared group*. The cardinality constraints of the roles of R are computed from the cardinalities of the attribute in the highest level of the projection that defines the shared group. When the cardinalities of a role cannot be

[5] In the future, less strict relationships will be considered, such IS-A relations.

computed, the most general one is assumed, i.e., [0–N]. The stereotype[6] "?" indicates that it must be validated and further refined.

If the transformation removes a compound attribute which is a superset of the components of the shared group, the additional attributes are lost. For instance, attributes "Quantity" and "Total" of entity type "Shopping Cart" are not part of the *master group* in entity type "Product". Removing compound attribute "Chosen products" induces the loss of attributes "Quantity" and "Total". We propose two transformations to keep these attributes.

2. If they appear to be characteristics of the relationship type between the entity types, they are moved to this relationship type.
3. Otherwise, they are moved to the *master* entity type. Since attributes "Quantity" and "Total" are only meaningful for the "Product" in the context of the "Shopping Cart", they are moved to the newly created relationship type.

3.5 Remarks on the Completeness of the Resulting Conceptual Model

The resulting conceptual model can be incomplete and needs to be validated by the analyst. For instance, some role cardinality constraints have been left undefined during conceptualisation. Moreover, some elements extracted from the logical user interface model actually are derived or computed attributes. These attributes appear in the resulting conceptual model but need not be made persistent and can be discarded.

This emphasises the fact that user interfaces analysis, despite its importance, must be complemented by other, more traditional, information sources when required. User interviews and legacy application observation and analysis are popular techniques for requirement elicitation that can be used to validate and complete the conceptual schema obtained so far.

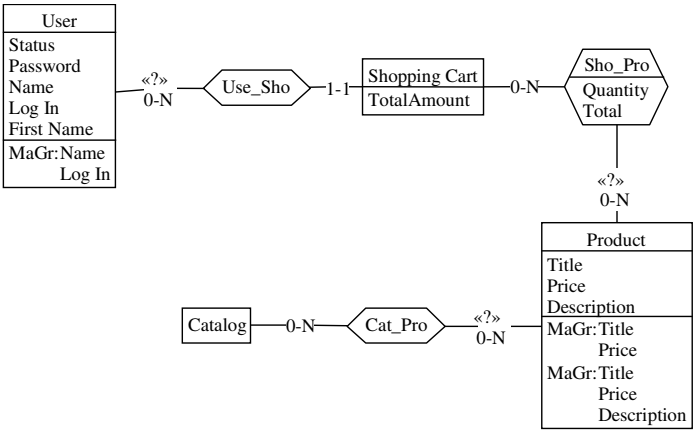

Fig. 7. Partially conceptualised data structures

[6] GER stereotype has the same meaning (or absence thereof) as UML's.

4 Conclusions

The process described in this paper is one of the components of the ReQuest framework, the goal of which is to develop a comprehensive environment to specify and generate in a semi-automated way data-intensive applications.

Our contribution to database design in complex web-based application development is threefold. First, it allows a more natural and more intensive user involvement[7] in requirement definition. Second, it gives form-based analysis a clean 3-step architecture derived from database reverse engineering methodologies: (1) physical structure extraction, through XHTML and XUL form parsing, (2) structural enrichment by eliciting shared subtrees through tree mining and normalisation techniques and (3) semantic interpretation or - conceptualisation - of the enriched structures through transformational techniques. Third, specific tools, integrated in the DB-MAIN environment, that support the three steps of this methodology have been developed. In this way, form-based analysis can be integrated in more general tool-supported database design methodologies.

Three case studies have been designed to validate the ReQuest framework, including the requirement elicitation and analysis process. The first one is a *training session management system* with 10 user interface fragments and a schema of 15 entity types. The second one is a *real estate advertising platform* for which 10 user interface fragments have been drawn. The resulting conceptual schema comprises 21 entity types. The last one is a classical *web sales platform* that comprises 20 user interface fragments translating into 4 entity types. Though these studies are recent and limited, we already have drawn interesting information.

1. *Size estimation*. There is no relation between the number of interface fragments and the size of the schema.
2. *Scalability*. The approach is more scalable than expected, but for unexpected reason! Indeed, it appears that a subsystem of 15-30 interface fragments constitutes the ideal work package. This size is manageable in a short time by a motivated user (1-3 days), the result can be visually validated by an expert designer in 2-4 days and the algorithms that extract the redundant substructures and derive the conceptual schemas are still fast despite their complexity (1-5 seconds for *FreqT*). Moreover, the resulting conceptual structures are fairly stable, that is, additional input interface fragments do not significantly improve the quality of the result. Larger systems can be split into subsystems whose conceptual schemas are then integrated through traditional methods [28].
3. *Completeness*. For each case study, the result has been compared with a reference schema elaborated with other methods. It appears that the only missing concepts are processed by well identified non-interactive workflows. Therefore the user model was complete in all cases.

[7] The idea to imply end users in the specification of web-based application (e.g., a business-to-customer application) may seem somewhat paradoxical since the real users (the hopefully thousands of customers) cannot be asked to draw their preferred interface. In such cases, representative corporate employees, for instance from the marketing department, can be substituted for these *undefinable* users.

4. *Soundness*. Some concepts were absent from the reference schema. They all were derivable attributes that were erroneously considered basic data by the user. Helping him/her to refine the definition of the fields quickly solved the problems.
5. *Quality*. The resulting schemas were as good as unnormalised schemas extracted through standard techniques. For instance, some ISA relations were left in their physical translation (one-to-one relationship type for instance) and needed further processing. This process is standard and is not specific to our approach.
6. *Acceptance*. The users enjoined being involved in the building of their future tool. In addition, the resulting schema were quite well accepted, since it merely translated in another formalism the information requirements of the users.

Future work will mainly be devoted to (1) enrich the conceptual schema, e.g., by exploiting the dynamic links and information transfer between forms, (2) enhance the validation process through paraphrasing and prototyping techniques, (3) introduce induction reasoning to exploit sample data provided by users.

References

1. The British Computer Society & Royal Academy of Engineering: *The Challenges of Complex IT Projects*. Published by the Royal Academy of Engineering (2004)
2. Damas, C., Lambeau, B., Dupont, P., van Lamsweerde, A. : Generating Annotated Behavior Models from End-User Scenarios, to appear in *IEEE Transactions on Software Engineering*, Special Issue on Interaction and State-based Modeling, 2006.
3. Brogneaux, A-F., Ramdoyal, R., Vilz, J., Hainaut, J.-L.: Deriving User-requirements From Human-Computer Interfaces, in *Proc. of 23rd IASTED Int. Conf.*, Innsbruck, Austria, Feb. 2005.
4. Thiran Ph., Hainaut, J-L., Houben, G-J., Benslimane, D.: Wrapper-based Evolution of Legacy Information Systems, to appear in *ACM Transactions on Software Engineering and Methodology* (TOSEM), 2006.
5. Schewe, K. D., Thalheim, B.: Conceptual modelling of web information systems. *Data Knowl. Eng*. 54(2): 147-188 (2005)
6. Batini, C., Ceri, S., & Navathe, S., B. *Conceptual Database Design*, Benjamin/Cummings. 1992
7. Coyette, J. Vanderdonckt, A.: Sketching Tool for Designing Anyuser Anyplatform Anyplatform, Anywhere User Interfaces, in *Proc. of the 10th IFIP TC 13 Conf. On Human Computer Interaction Interact '05*, Roma, Italy, September 2005.
8. Hainaut, J.-L.: *Introduction to Database Reverse Engineering*, 3rd Edition, LIBD Publish., Namur, 2002 [http://www.info.fundp.ac.be/~dbm/publication/2002/DBRE-2002.pdf]
9. Hainaut, J.-L.: Transformation-based Database Engineering, Chapter in *Transformation of Knowledge, Information and Data: Theory and Applications*, P. van Bommel Editor, IDEA Group (2005).
10. Rollinson, S. R., Roberts, S. A.: Formalizing the Informational Content of Database User Interfaces. Conceptual Modeling - ER'98, in *Proc. of the 17th International Conference on Conceptual Modeling*, Springer-Verlag, (1998) 65-77
11. Choobineh, J., Mannino, M. V., Tseng, V. P.: A Form-Based Approach for Database Analysis and Design. *Com. of the ACM*, Vol. 35, N°2, (February 1992) 108-120
12. Lee, H., Yoo, C.: A Form-driven Object-oriented Reverse Engineering Methodology, *Information Systems*, Vol. 25, No. 3, Elsevier, 2000

13. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing, Web sites. WWW9/Computer Networks 33(1-6): 137-157 (2000)
14. Conallen, J.: Building Web Applications with UML. Addison Wesley (Object Technology Series), 2000.
15. Oaks, P., ter Hofstede, A., H., M., Edmond, D. , Spork, M.: Extending conceptual models for web based applications, in *Proc. of ER Conference 2003*, Springer
16. Brambilla, M.: Extending hypertext conceptual models with process-oriented primitives, in *Proc. ER Conference* 2003, Springer
17. *WebML*, [http://webml.org/]
18. Cato, J.: User-Centered Web Design, Addison-Wesley, 2001.
19. Escalona, M., Koch, N., Requirements Engineering for Web Applications – A Comparative Study, in *Web Engineering Journal*, Vol. 2, No. 3, 2004
20. Jakob Nielsen : User Interface directions for the Web, in *Com. of the ACM*, Volume 42, Number 1 (1999), pp 65-72.
21. Gibson, D., Punera, K., Tomkins, A. : *The Volume and Evolution of Web Page Templates*, David Gibson Center (2005).
22. Collopy, E., Levene, M., Evolving Example Relations to Satisfy Functional Dependencies, in *Issues and Applications of Database Technology*, 1998, pp. 440-447.
23. Chi, Y., Nijssen, Y., Muntz, R., Frequent Subtree Mining - An Overview. *Fundamenta Informaticae* XXI. IOS Press (2001) 1001–1038
24. Yang, L., Lee, M., Hsu, W., Guo, X.: 2PXMiner - An Efficient Two Pass Mining of Frequent XML Query Patterns. In *Proc. of the SIGKDD2004*. 2004.
25. Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., Arikawa, S.: Efficient Sub-structure Discovery from Large Semi-structured Data. In *Proc. of the 2nd Annual SIAM Symposium on Data Mining*, 2002.
26. Cohen, W. W., Ravikumar, P., Fienberg, S. E.: A comparison of string distance metrics for name-matching tasks. In *Proc. of the IJCAI-2003*. 2003
27. Hainaut, J.-L.: Entity-Relationship models: formal specification and comparison, In *Proc. of the 9th Int. Conf. on the Entity-Relationship Approach*, 1991.
28. Spaccapietra, S., Parent, C., Dupont,Y.: Model independent assertions for integration of heterogeneous schemas. in *VLDB Journal*, 1 (1992) 81–126