Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2006 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2006

Inferring the User Interface from an EER Data Schema

Akhilesh Bajaj
University of Tulsa

Inferring the User Interface from an EER Data Schema

Akhilesh Bajaj
University of Tulsa
akhilesh-bajaj@utulsa.edu

ABSTRACT

Much of the work on automatic user interface (UI) generation has met with limited success because of the added load on the human designer to use specialized scripts for UI specification. In this research in progress, we propose a methodology applicable to database driven systems that a) automatically infers a draft interface directly from an extended entity relationship (EER) model schema and b) lists the interactions that need to take place between the designer and the tool in order to generate the final user schema.

Keywords

user interface, data model, automatic generation, screens, graphical user interface

INTRODUCTION

The graphical user interface has become both ubiquitous and relatively uniform in providing access to applications for diverse users (Myers et al., 2000). From the early 1980-s, user interface (UI) management systems focused on providing human designers high-level specification languages such as state transition diagrams or event based representations to specify the interface in response to events (Jacob, 1986, Olsen, 1986). These representations have become progressively richer and model-based interface development tools today range from automatic interface generators to tools that offer advice based on task representations. In (Szekely, 1996) model based tools were classified into five basic categories: a) automatic interface generator (AIG), b) specification based toolkits that do not automatically generate interfaces but provide a specification language that expresses design, c) help generation toolkits that generate documentation, d) modeling tools that allow visual representations of interfaces before they are built and e) advisory tools that provide advice to designers. In this work, we focus on combining the best aspects of AIG and collaboration methodologies, in order to develop interfaces to database applications.

Figure 1 (Szekely, 1996) describes the model-based interface development process. The model component organizes the specification into three layers. Domain models correspond to the data schema. Examples of task models include data flow diagrams or other activity diagrams. An abstract UI specification provides a set of low level interface tasks such as selecting from a set of elements, information elements selected from the domain model, and how the two should be grouped. The concrete UI specification deals with the actual interface elements such as the windows, buttons, checkboxes and navigation buttons.

Based on figure 1, it is clear that the majority of model-based environments explicitly differentiate between task (process) models and data models. Initial AIG toolkits like JANUS (Balzert et al., 1996) used only data models and attempted to derive user interfaces from these models. This approach had the advantage of not requiring any additional work other than the creation of the data model. However, there were several problems with this approach, as pointed out in (Harning, 1996):

a) The UI may need information from different tables on the same screen.

b) Users may need summarized information, not just the raw data from the tables.

c) The interface may be more effective with information displayed in the form of plots or charts, not just alphanumeric information.

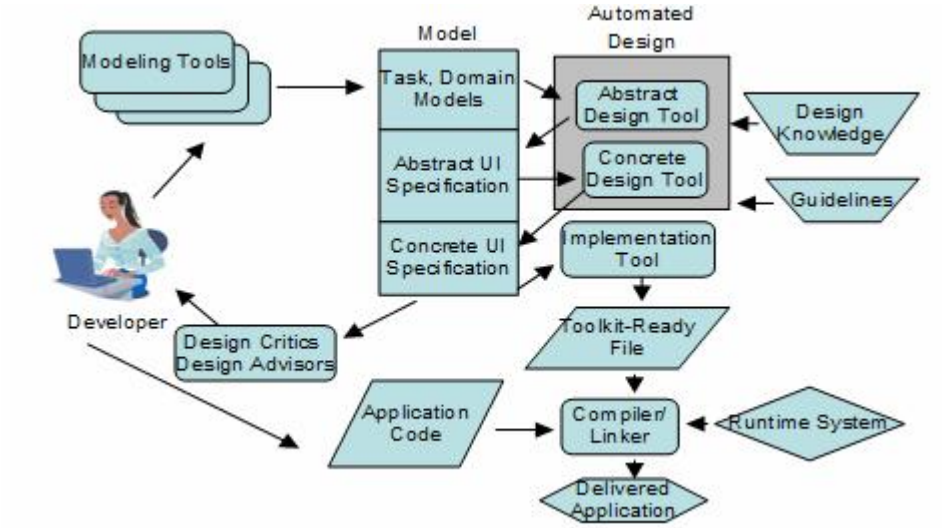

Figure 1. Model-Based Interface Development Process (Szekely, 1996)

One solution to the problems highlighted above is to explicitly require the designer to describe the information present on each screen, along with navigation patterns, etc. This is the approach used in tools such as TRIDENT (Bodart et al., 1995) and TADEUS (Stary, 2000). However, *this approach also has a significant disadvantage*: it places greater burden on the designer who now has to recapture the user requirements another language, and simultaneously ensure that all the items in the domain model (data) are represented somewhere in the new specification. We argue that this is one reason why AIG toolkits have not met with much acceptance in the designer community, and why most interfaces are still designed manually (Myers et al., 2000).

In this work, we attempt to balance the advantages of both approaches with regard to business applications. The very great majority of business applications involve a database back-end with a front-end UI, and hence we utilize the extended entity relationship (EER) model to capture the data schema (Chen, 1976, Smith and Smith, 1977). Our methodology uses a set of rules to map EER objects automatically to provide a first cut user-interface, and then provides an opportunity for a structured dialog with the user to attempt to assuage some of the problems with the data-model-only approach.

A DETAILED METHODOLOGY TO INFER A UI FROM AN EER SCHEMA

As summarized in (Szekely, 1996), the basic steps in an AIG algorithm consist of determining:

a) *P* **:**presentation units (the different windows and a list of their contents)

b) *N***:** navigation between presentation units

c) *A***:** abstract interface items (*e.g.*, a drop down list, a check box, etc.)

d) *C***:** concrete interface items (how each abstract item will be implemented, such as a list-of-values for a drop-down list specification)

e) *L***:** window layout (position, font size, and other presentation criteria

Before presenting the methodology, we list the concepts in the EER model that we will map. We base our list of concepts largely from standard database textbooks like (Korth et al., 2005) and assume an EER schema to consist of the following concepts:

-entity sets,

-relationship sets with *0/1* cardinality on at least one side, and any cardinality on the other,

-relationship sets with *m:n* cardinality and *n*-ary relationships

-attributes of entity and relationship sets,

-multi-valued attributes

-entity subclasses that have extra attributes and/or extra relationships, with no multiple inheritance

-weak entity sets (existence dependencies) with a unique identifier

Our methodology, described next, consists of two phases: the *automated generation* of the first-cut interface (FCI), followed by the *structured dialog with the human designer* to generate the second-cut interface (SCI).

Generating the FCI Schema

For convenience, we present our methodology for FCI generation in the following format. For each EER concept, we list the mapping to a relational schema and to the UI. We will not consider the *C* and *L* components of the mapping here, since *C* is system dependent (*e.g.*, different UI systems will implement drop down lists differently) and *L* is beyond the scope of this paper (L would typically include color choice, font selection, text alignment, etc).

Entity Sets

Relational mapping: Create a table for the entity set. The columns of the table are the attributes of the strong entity set. The primary key of the table is the primary key of the entity set.

UI Mapping:

P: Create a separate screen with all the attributes of the entity set. Add update, insert and delete buttons that allow basic database operations.

N: The screen gets links to the screens that correspond to every *m:n* relationship set in which the entity set participates, and to the screens that correspond to every multi-valued attribute of the entity set

A: -For enumerated type attributes, provide a fixed list of values.

-For attributes that are dates, currency, strings or numbers , provide a text box.

-For the primary key attribute(s), provide a non-updatable text box

-For attributes that are Boolean, provide a check box

Relationship sets with 0/1 *cardinality on at least one side, and any cardinality on the other*

Relational mapping: Add a column(s) in the table that corresponds to the entity set that is on the "Any Cardinality" side. The column(s) we add here is the primary key of the entity set that is on the 0/1 side and any other attributes of the relationship set.

UI Mapping:

P: Use the screen developed for the entity set that corresponds to the "Any Cardinality" side

N: No additional navigation provided here

A: -Provide a drop-down list of values that show the primary key of the entity on the "Any Cardinality" side.

-Follow the same rules for other relationship attributes as described for entity sets.

Relationship sets with m:n *cardinality and* n-*ary relationships*

Relational mapping: Create a separate table for the relationship set. The columns of the table are the attributes of the relationship set (if any) + primary keys of all the entity sets that participate in the relationship set. The primary key of the table is = the primary keys of all the entity sets that participate in the relationship set.

UI Mapping:

P: Create a separate screen with all the attributes of the relationship set, as well as the primary keys of all participant entity sets. Add update, insert and delete buttons that allow basic database operations. If the relationship has no attributes then disable the update button.

N: The screen gets links to the screens that correspond to every participant entity set

A: -For enumerated type attributes, provide a fixed list of values.

-For attributes that are dates, currency, strings or numbers , provide a text box.

-For the primary key attributes, provide a drop-down list of relevant values drawn from the participant entity sets

-For attributes that are Boolean, provide a check box

Multi-valued attributes

Relational mapping: Create a separate table for the multi-valued attribute. The columns of the table are the primary key of the entity set to which the attribute belongs + a separate column for values of the attribute. The primary key of the table is all the columns of the table.

UI Mapping:

P: Create a separate screen with the primary key of the entity set and the multi-valued attribute. Add update, insert and delete buttons that allow basic database operations.

N: The screen gets a link to the screen for the entity set that owns the multi-valued attribute.

A: -For enumerated type attributes, provide a fixed list of values.

-For attributes that are dates, currency, strings or numbers, provide a text box.

- For the primary key attributes, provide a drop-down list of relevant values drawn from the owner entity set

Entity subclasses that have extra attributes and/or extra relationships

Relational mapping: Create a separate table for the superclass first, using the rules for mapping entity sets we have seen earlier. For each subclass entity set, create a separate table. The columns of each table = the additional attributes of the corresponding subclass entity set + the primary key of the superclass entity set. The primary key of the subclass table is the primary key of the superclass table.

UI Mapping:

P: Create a separate screen with all the extra attributes of the subclass, as well as the primary key of the superclass entity set. Add update, insert and delete buttons that allow basic database operations.

N: The screen gets a link to the screen that corresponds to the superclass entity set

A: -For enumerated type attributes, provide a fixed list of values.

-For attributes that are dates, currency, strings or numbers , provide a text box.

-For the primary key attributes, provide a drop-down list of relevant values drawn from the superclass entity sets

-For attributes that are Boolean, provide a check box

Weak entity sets (existence dependencies) with a unique identifier

Relational mapping: Create a separate table for the weak entity set. The columns of the table are the attributes of the weak entity set + the primary key of the corresponding strong entity set. The primary key of the table is the primary key of the corresponding strong entity set + the unique identifier of the weak entity set.

UI Mapping:

P: Create a separate screen with all the attributes of the weak entity set, as well as the primary key of the strong entity set. Add update, insert and delete buttons that allow basic database operations.

N: The screen gets a link to the screen that corresponds to the strong entity set

A: -For enumerated type attributes, provide a fixed list of values.

-For attributes that are dates, currency, strings or numbers , provide a text box.

-For the primary key attributes, provide a drop-down list of relevant values drawn from the superclass entity sets as well as a non-updateable field for the unique identifier attributes of the weak entity set.

-For attributes that are Boolean, provide a check box

Generating the SCI Schema

For reasons of space we do not present this step in detail. The motivation is to overcome some of the earlier disadvantages of AIG toolkits. Thus, the steps include designer feedback to remove attributes from certain screens (*e.g.* salary from the employees screen), add query fields that fetch raw or summarized data from other tables, add graphic reporting and add more intuitive identifiers such as *name* to the primary keys in the drop down lists. In order to simplify the human workload, we do not allow the addition of any updateable fields, the idea being that each screen provides at most one updateable table, though multiple readable tables. This is similar to the notion of updateable views in the database literature.

AN ILLUSTRATIVE EXAMPLE OF A FIRST CUT INTERFACE

Figure 2 depicts a simple EER schema, following standard diagramming conventions (Korth et al., 2005). Attributes are next to each entity and relationship set. Figure 3 illustrates the 4 screens in the FCI, generated using the AIG algorithm outlined above.

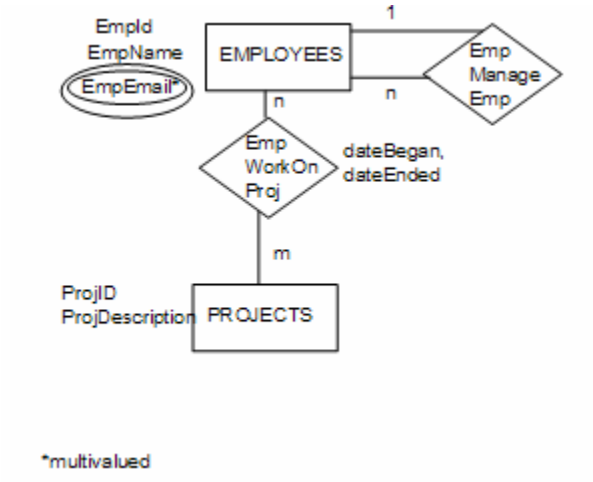

Figure 2. EER Schema for Application

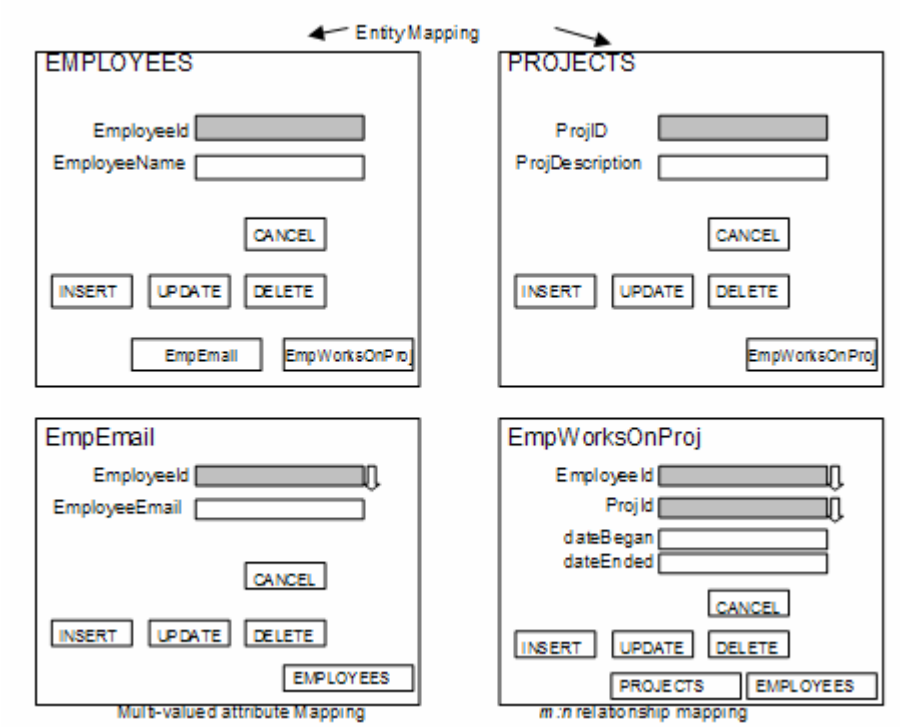

Figure 3. The Four First Cut UI Screens from the EER Schema

We note that this first cut interface *will then be processed by the designer as described earlier, to produce the SCI,* which can then be evaluated by end-users.

CONCLUSION

In this research in progress, we propose a methodology to infer a UI from an EER schema. The chief contribution of the methodology is that it focuses on database driven applications, and balances automatic generation of the UI with input from the designer in order to arrive at a final UI. As part of this work, we aim to extend the rules described here to incorporate higher level navigation screens, construct a compiler that automatically generates the first cut schema based on the rules described here, and test the methodology for large scale applications.

REFERENCES

1. Balzert, H., Hofmann, F., Kruschinski, V. and Niemann, C. (1996) In *Proceedings of CADUI '96*(Ed, Vanderdonckt, J.) Presses Universitaires de Namur, Namur, Belgium, pp. 183-206.
2. Bodart, F., Hennebert, A., Leheureaux, J., Provot, I., Sacre, B. and Vanderdonckt, J. (1995) In *Design, Specification and Verification of Interactive Systems*Springer, Vienna, Austria, pp. 262-278.
3. Chen, P. P. (1976) *ACM Transactions on Database Systems,* **1,** 9-36.
4. Harning, M. (1996) In *Proceedings of CADUI'96*(Ed, Vanderdonckt, J.) Presses Universitaires de Namur, Namur, Belgium, pp. 121-138.
5. Jacob, R. J. K. (1986) *ACM Transactions on Graphics,* **5,** 283-317.
6. Korth, H., Silberschatz, A. and Sudarshan, S. (2005) *Database Systems Concepts,* McGraw Hill, New York.
7. Myers, B., Hudson, S. E. and Pausch, R. (2000) *ACM Transactions on Computer-Human Interaction,* **7,** 3-28.
8. Olsen, D. R. (1986) *ACM Transactions on Information Systems,* **5,** 318-344.
9. Smith, J. M. and Smith, D. C. P. (1977) *ACM Transactions on Database Systems,* **2,** 105-133.
10. Stary, C. (2000) *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans,* **30,** 509-525.
11. Szekely, P. A. (1996) In *Design, Specification and Verification of Interactive Systems: Proceedings of the Third International Eurographics Workshop*
12. (Eds, Bodart, F. and Vanderdonckt, J.) Namur, Belgium.