UI Generation from Task, Domain and User models: the DB-USE Approach

Vi Tran
Louvain School of Management-PRISME, Université catholique de Louvain
Louvain-la-Neuve, Belgium
Vi.Tran@uclouvain.be

ABSTRACT
Information Systems UI (User Interface) generation from declarative models has been the focus of numerous and various approaches in the human computer interaction community. Typically, the different approaches use the different models based on their singular aspects. This paper proposes a new process that combines the task, domain, and user models taken together to drive the information system user interface design and code behind generation. To this end, we propose a framework, i.e., a methodological process, a meta-model and a software prototype called DB-USE.

Keywords
Task Model, Domain Model, User Model, Automatic User Interface Generation

INTRODUCTION
Numerous research works in the area of UI generation have focused on UI generated from the declarative models such as Tood[2], Trident [1], Teallach[3], Goliath [4]. Declarative models can provide a more abstract description of the UI and facilitate the creation of processes to design and implement the UI in a systematic way since they offer capabilities [5].

Most researchers have richly described and detailed the automatic user interface generation; they forgot that the application code generation for performing the functions of an application is also important, specially, the generic functions of a database application for an information system such as AddNew(), Display(), Update(), Delete(), Search(), Review() functions.

This paper focuses on three models (task, domain and user) for generating UIs:

- The task model describes the abstract user interface. It expresses how an end user may want to interact with a system in order to reach a given goal. This expression is intended to be independent of any particular implementation or technology. This explains why a same set of models could initiate several different user interfaces [6].
- The domain model provides the special features for creating a user interface and specifies the generic application functions. These features are the attributes of the objects in the domain model and the relationships between these objects.
- The user model supports the creation of user interfaces taking into consideration the preference of the users.

We aim at defining and using a methodological framework for developing UIs to information systems/database applications from the task model, domain model and user model combined together. This framework consists of :

- A meta-model that governs the semantics of the task, domain and user models.
- A methodological process that supports the UI designer/developer to create the UIs semi-automatically from the three models.
- A software prototype supporting the methodological process in order to generate both the user interface code and the application code that performs the basic functions of a database application.

DB-USE: METHODOLOGY
DB-USE (DataBase User interface System Engineering) is specifically interesting to the benefit of the UI designer. It is built to support them as much as possible in every phases of the UI generation process. In DB-USE, this process consists of five phases: model analysis, relation making, UI design, application function design, and code generation. One person is responsible for each phase.

The **Model analyst** is responsible for the model analysis phase. The task, domain and user models are automatically built by the DB-USE system from the existing resources. The **User Model Editor** is used to load the user model from a diagram file; the **Task Model Editor** is used to load the task model from a XML file; the domain model is loaded from a concrete database by the **Task Model Editor**. The generic functions of a database application which are defined by DB-USE are included into its domain model. These three components are implemented independently. Once these models have been loaded, they are verified and modified by the **Model analyst**. He can change the attribute of the tasks such as name, type or delete an existing task or add a new one.

The associations between the components of the task and domain models are made by the **UI Developer** in the relation making phase. DB-USE's task model includes two types of task: operation and action tasks. The operation tasks are linked to the domain's attributes and the action tasks are linked to the defined functions. The **UI**

Developer selects the tasks and the domain's attributes which participate in the associations.

In the UI design phase of DB-USE process, the CIOs (Concrete Interaction Objects) are created by the **UI Builder** based the associations made in the relation making phase and the user model. The user model is used to generate individual user interfaces and to select the control type among the possible ones that matches the preference of the end-users. Once the programming language has been determined, the FIOs (Final Interaction Object) are created based on the CIOs. In order to specify the attributes of the UI objects, DB-USE uses the **Mapping rules [9]** base which defines the rules for making the associations between the tasks and domain's components, specifying the control type, location, dimension. In this phase, the **UI Designer** plays an important role, he has to make sure that the created UI objects are correct with respect to the goal of the tasks. On the other hand, he can change the attributes of these objects such as control type, label, dimension…

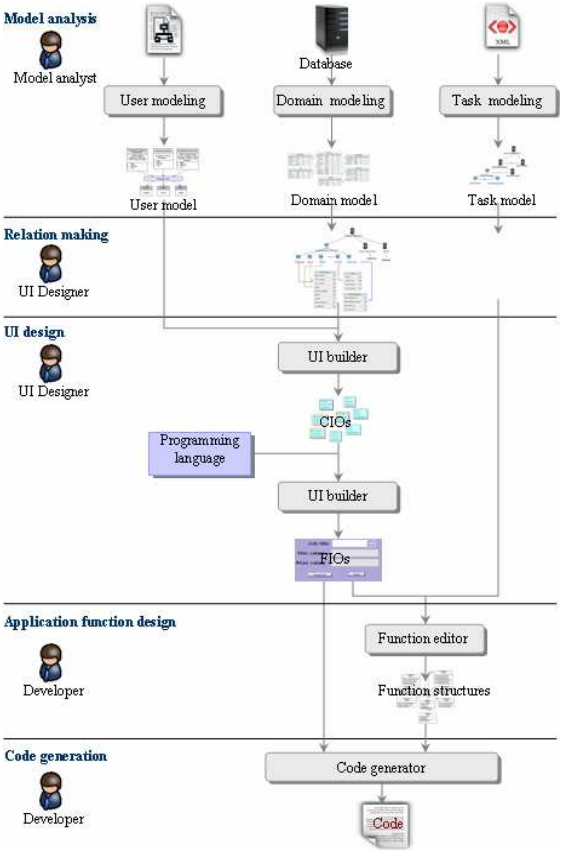

Figure 1: The methodological process of DB-USE

The application function design phase is started once the FIOs have been completely created. The **Function** will determine the linked functions of the created FIOs to construct the structure for these functions (Display(), AddNew(), Update(), Delete(), Search(), Review(), …). One strong advantage of DB-USE is that application functions are automatically constructed by the system not by the **Developer**. But the **Developer** can modify the structure of the functions, define a new function, delete an existing one.

Finally, the UI code and the application code are generated in the code generation phase by the **Code generator**. The role of the **Developer** in this phase is to verify the generated code then modify it if it is needed in order to ensure it is successfully compiled. Typically, the code generated by DB-USE is clear and complete enough to be compiled and executed immediately.

Loading Task, Domain and User Models

Loading Task Model

DB-USE's task model is loaded from a XML file which is built by CTTE (ConcurTaskTrees Environment) [7]. As already said, this model implies two types of tasks:

Action tasks are used to describe the end-user's command to the system such as close a dialog, delete a data record, search information, open a dialog and so on.

Operation tasks are used to describe the display of information to end-user or the reception of the information from the end-user.

The defined types of tasks at the design level are action and operation types. The types of tasks in CTTE, read from the XML file, are abstraction, interaction, cooperation, application and user are typically defined at the analyst level. At the analyst level, the defined tasks represent different kinds of information including unnecessary ones for UI generation. For instance, the user task is a cognitive task that the end user selects as a strategy to solve a problem or checks the result; typically this type of task is not used to generate the user interface. Therefore, the user task is translated to none and the others are translated to action or operation tasks at the design level based on the Table 1.

Task mapping	
Abstraction task	Action task
Interaction task	Action task/ Operation task
Cooperation task	Action task
Application task	Operation task
User task	None

Table 1: Task mapping table

Figure 2 depicts an example of the mapping between tasks in ConcurTaskTrees and in DB-USE considering a typical *AccessStudent* Data task.

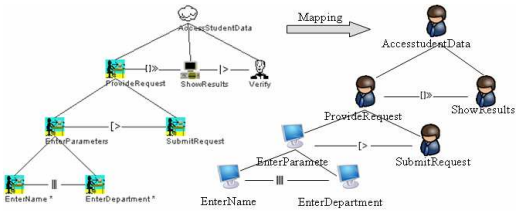

Figure 2: Transforming CTTE's tasks to DB-USE's tasks.

Loading Domain Model

DB-USE's domain model is loaded from a concrete database which is specified by the UI developer. This model is a representation of both the objects in a domain and their relationships. The information in the domain model is basically the data model where the data objects are defined including the relationships between the data objects, and other information that is pertinent to the relationships such as business logic. The domain's object of DB-USE is described by the object's name and its attributes including name, data-type, is-null, primary-key (See Figure 4).

Moreover, the generic functions of a database application are also built in this model. Unlike current applications such as Teallach[3] and Goliath[4] in which these functions are defined by the UI developers, in DB-USE they are automatically defined and constructed.

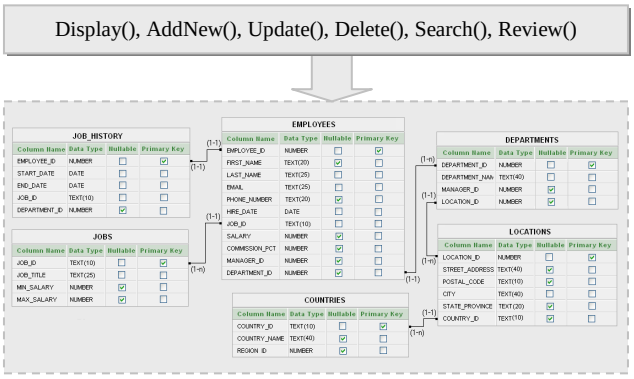

Figure 3: Domain model.

Loading User Model

The end-users are grouped into three groups named Complex, Mean and Simple based on the information from the user model such as preferences, capabilities, psychomotor skills, characteristics, … of the end-user.

Making Links

Once the task, domain and user models have been loaded, the UI developer will make the associations between the tasks and the domain's attributes, the defined functions by linking them together. The operation tasks are linked to the domain's attributes; the action tasks are linked to the defined functions as depicted in Figure 4. One action task is linked to only one defined function; but one operation task can be linked to more than one domain's attribute and, one attribute can be linked to more than one task.

Creating the UI Objects

Firstly, we present two interaction objects that we use in our process: concrete and final interaction objects.

A **Concrete Interaction Object** (CIO) is a graphical object for entering and displaying the data that the user can see, feel and manipulate [8]. A CIO is structured by its label, its control type and its editable status.

Final Interface Object (FIO) represents the operational interface object that is running on a special computing platform either by interpretation or by execution. The FIO is determined based on the CIO in a certain language, on a certain platform. A FIO is structured by the label, the control type, the editable, the location, the dimension.

The CIOs are created based on the links made in the previous step and our RSD (Rules for Specifying Data-type), RGA (Rules for Grouping Attributes), and RDC (Rules for Determining control types) rules. These rules are discussed in[9]. Besides using the rules above to determine the control type of the CIOs, the ***User model*** is also used to determine the control type which matches the preference of the end-users. Based on the three options including **simple, mean** and **complex** options, the system selects a correct control type among the possible control types. (Figure 4)

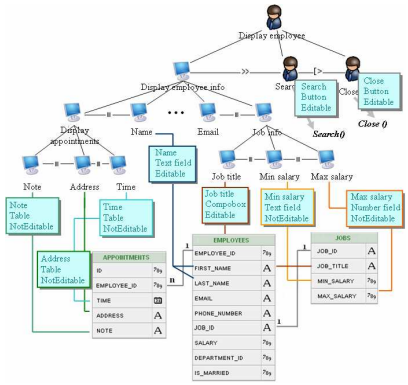

Figure 4: Creating CIOs based on the links between tasks and domain's attributes, defined functions.

Once CIOs have been created, they are transformed to FIOs based on the programming language which is determined by the developer.

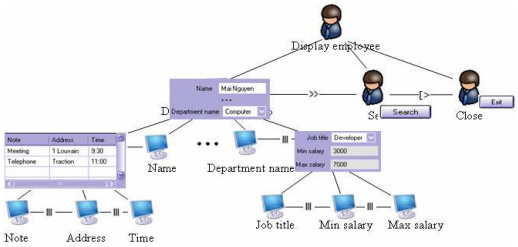

Figure 5: Transforming CIOs to FIOs.

Figure 5 depicts the FIOs transformed from CIOs depicted in Figure 4.

Generating the UI and Application Codes

Unlike the current applications, DB-USE generates both the user interface code and the application code. The generated code is a complete code which can be compiled and ran immediately when usually code generated by the other applications are just prototypes or code skeletons.

DB-USE TOOL

DB-USE has been implemented in Java. Java is widely used in the large database-oriented software packages such as Oracle or SAP. Besides, Java is platform independent.

The main purpose of DB-USE is to help designers to create the user interface and generate the application code for

performing the generic functions of a database application easily and quickly. We summarize below the important goals DB-USE aims to achieve:

***A friendly and simple interface*:** The interface of DB-USE has been designed in order that any designer can work with it giving her/him the important features mentioned above. Similarly to other applications, the main interest of our tool is the visual and/or graphical generated result.

***Performing most of the functions discussed in the domain model*:** DB-USE has been developed to perform most of functions defined in the domain model. These functions makes the user interface directly specified based on the associations between the task and domain models; and the capability of the application code generation.

***Managing efficiently existing resources*:** As already discussed, the task, user and domain models used in our process are the existing resources; they are used by the other processes such the analyst process.

***Visualizing the process*:** The interface visually supports our process as much as possible. Currently, the tasks are displayed in terms of graphical objects. The designer can review the result in terms of graphics whenever he wants during working.

Generating and running the code: In the current version, the user interface and application codes generated by DB-USE are also implemented in Java. The code generated has been made clear, easy to understand and documented. The generated code can be immediately compiled by a java compiler without modifications.

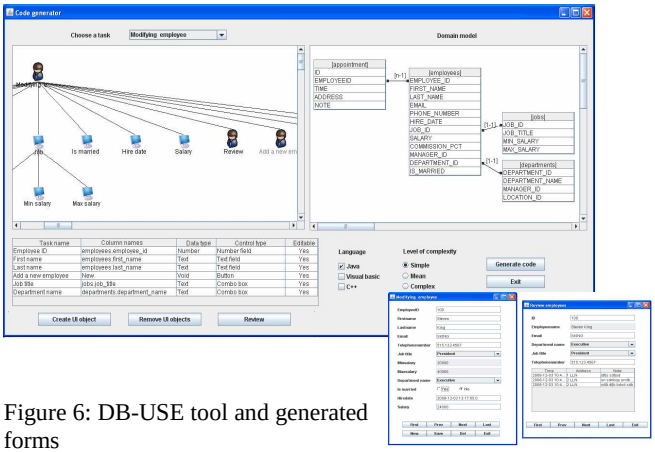

Figure 6: DB-USE tool and generated forms

Figure 6 depicts DB-USE. The tool is divided into three separate areas. The first area is used to display the task model; the second the domain model and the third one the information of the UI objects.

DB-USE is especially interesting to represent the generated code. In order to help the designer to maintain, enhance and develop the code, the generated code should be identified, clear and easy to understand. The user interface code generation and the application code generation are separately presented.

CONCLUSION

To be efficient, information-intensive systems that are an important component of today's software applications need effective human-computer interaction. User interfaces for such data systems has been a recurrent research issue and nowadays these UI have to support automatic generation to adequately be dealt with.

This framework has aimed at offering a low cost, short time-to-implementation and efficient UI development environment from the business user side. Indeed, the objective of this tool has not only been to generate the user interfaces but also to generate the code for performing the generic functions of the database applications.

This research has also contributed to define rules for mapping task and domains models to one another in both ways and translate these models into code in order to automate the user interface design process.

REFERENCES

1. Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Sacre, B., Vanderdonckt, J., Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide. *Interactive Systems: Design, Specification and Verification*. Springer, 1995, p. 77-94
2. Mahfoudhi, A., Abed, M., Abid, M., Towards a User Interface Generation Approach Based on Object Oriented Design and Task Model. *TAMODIA'2005 : 4th International Workshop on TAsk MOdels and DIAgrams for user interface design For Work and Beyond Gdansk*, Poland September 26-27, 2005.
3. Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J., Gray, P.D., Cooper, R., Goble, C.A., Pinheiro da Silva, P., Teallach: a model-based user interface development environment for object databases. *Interacting with Computers*, Vol. 4, No. 1, December 2001, pp. 31-68.
4. Julien, D., Ziane, M., Guessoum, Z., Goliath: An extensible model-based environment to develop user interfaces. *In Proc. of the Fourth In. Conference on Computer Aided Design for User Interfaces*, Kluwer Academics Publishers, 2004.
5. Pinheiro da Silva, P., User Interface Declarative Models and Development Environments: A Survey. *In Proceedings of DSV-IS2000*, volume 1946 of LNCS, p. 207-226, Limerick, Ireland, 2000. Springer-Verlag.
6. Wilson, S., Johnson, P., Bridging the generation gap: From work tasks to user interface designs. *In Computer-Aided Design of User Interfaces*. Namur University Press, 1996.
7. Paternò, F., Mori, G., Galiberti, R., CTTE: an environment for analysis and development of task models of cooperative applications. *In CHI '01 Extended Abstracts on Human Factors in Computer Systems*. Seattle, Mar, ACM Press.
8. Vanderdonckt, J., Bodart, F., Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. *In: Proc. of the ACM Conf. on Human Factors in* Computing Systems. ACM Press, New York (1993).
9. Tran,V., Vanderdonckt, J., Kolp, M., Faulkner, S., Generating User Interface from Task, User and Domain Models, *in Proc. of the Second Int. Conf. on Advances in Human-oriented and Personalized Mechanisms*, Technologies and Services, 2009.