Generating User Interface for Information Applications from Task, Domain and User models with DB-USE

Vi Tran, Jean Vanderdonckt, Manuel Kolp, and Yves Wautelet
Louvain School of Management-PRISME
Université catholique de Louvain
Louvain-la-Neuve, Belgium
{Vi.Tran, Jean.Vanderdonckt, Manuel.Kolp, Yves.Wautelet}@uclouvain.be

ABSTRACT
Database 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 and exploiting 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. The main difference in our work from other ones is to contribute the mapping rules for creating the UI objects and to combine the three task, domain and user models to generate the code for performing both the UI and the generic functions of a database.

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

INTRODUCTION
Human-computer interactions are today managed through ergonomic and powerful database user interfaces (UI). These UI require to access and present data from different types of structures that reside typically in enterprise functional modules such as human resources, sales, marketing, accounting, finances, manufacturing or productions ERP packages [22]. Naturally database interfacing has been a technical and human interaction issue since the seventies [2, 3]. But, with the today's increasing use of very large databases and extended business applications such as ERP II, Customer Relationship Management, E-procurement, Reporting/Drill-Down or Data Mining systems, these user interfaces require dynamic automation and runtime generation to properly deal with on a large-scale.

UI researchers have richly discussed the capability and importance of automatic user interface generation and propose them as the core of visual-based development environments [4]. There are currently numerous and various approaches using different input materials: designs, patterns, architectures, declarative models, … In this set of techniques, an emerging method is the automatic UI generation from declarative models [5, 2, 6], inspired from Fourth Generation Languages code generation [5]. In practice, these models are high-level abstractions such as goal or task [7], presentation, dialogue [8] or interaction, domain [9] models on which we will focus in the research.

Typically, a strong motivation for database application UI generation is the unexploited potential of relationships between objects. This makes possible to create task-domain mapping rules that are in turn mapped onto containers in the UI model [21].

Advantages of Declarative Models in User Interface Generation
User interface is that subset of a software system that interacts with the user of the system. Since it is used to communicate between the end-users and the system's computer, making it a crucial element, the successfulness of a software system depends on, besides the design of its architecture, the user interface design [11]. End-users expect UIs to be easy to use, understand and give the most adequate result.

The development of successful interactive system requires a careful design of the user interface. The user interface affects the acceptation of the users to use this application or not. A good user interface design helps the users to work effectively and quickly. The first step is to take a step-wise approach by using a methodological UI derivation process based on the available resources such as task, domain, data, user, presentation, dialog models. This process is conducted by designing the user interface manually or semi-automatically or automatically. In this paper, we will approach the semi-automatic user interface generation.

Numerous research works in the area of automatic user interface generation [24] have focused mainly on UI generated from the declarative models. They present the following advantages [12] in regard to both the automatic user interface and code application generations:

- They can provide a more abstract description of the UI than the ones provided by the other UI development tools.

- They facilitate designing and implementing the UI in a systematic way since they offer capabilities such as modeling user interfaces using different levels of abstraction; incrementally refining the models; and re-using UI specifications.
- They provide the infrastructure required to automate tasks related to the UI design and implementation processes.

An important point in using declarative models for creating the user interface is that these models typically described at a conceptual level. So that, the user interface created based on these model can be implemented in the different ways on the different delivery platforms.

User Interface Generation from Combining Task, Domain and User models

The information used to build the user interface usually comes from a large and complex context in which the users work to complete their tasks. This context includes the users' characteristics (user model), their current domain of application (domain model often linked in the database world to the data or conceptual model), the tasks they commonly perform (task model), the platform they work on (platform model), the device they are currently using (device model) and so on. This context is not fixed for all the user interface generating processes since it depends on the different approaches. For example, in TRIDENT [7] the user interface is determined from the task and application models while in Mecano [9] the user interface is determined from the sole domain model.

The task, domain and user models are three important models based on which the user interface is easily built for the following reasons:

- *The task model describes the abstract user interface.* The task model expresses how a 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 [13].
- *The domain model provides the special features needed for creating a user interface and specifying the methods for performing the generic application functions*. These features are the attributes of the objects in domain model, the relationships between these objects and prototype of the generic application functions. The domain model is not used separate from other models to generate the user interface, it is combined to the task, presentation models [21].
- *The user model supports the creation of user interfaces which consider to the preferences of the users*. Like the domain model, the user model is not used separately from other models to generate the user interface since there are various aspects of the user interfaces that are adapted according to user models.

Hence, combining models is an important concept in user interface generation since the different models describe different aspects of the UI. For example, a user interface generated from a task model is expected to be a means by which the user can communicate with the system to accomplish his task. The user interface generated from a user model is expected to support the users based on their characteristics. The user interface generated from several different models carry many needed aspects of a user interface. Various research works have focused on such models to generate UIs. For instance, TOOD [14] uses the task and user models to generate the UI. MECANO [9] generates UI based on the domain model. TRIDENT [8] or FUSE [15] combines the task and domain models while SUPPLE combines the user and device models.

Unfortunately, the researchers who have focused on user interface generation have not investigated and studied the implementation and code generation of application tasks, especially, those generic one related to editing, inserting, deleting and searching data. This research focuses on the code application generation for the tasks of a data-oriented application coming from the following main points:

- The data manipulation demand is very high. It is repeated regularly in most database applications especially for common tasks such as editing, inserting, deleting and searching data.
- These functions are easily performed through receiving data from the user, displaying data to the user, executing the different SQL select, insert, update and delete queries with respect to the different user requirements.
- Typically, we aim at defining and using a methodological framework for developing a user interface to database application 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 task, domain and user 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 the database application.

The paper is structured as follows. Section 2 overviews the related work in model-based UI generation. Section 3 described the models used in our approach. Section 4 is concerned with the description of our process and Section 5 describes DB-USE in terms of CASE-Tool technology. Finally, we conclude the paper.

MODEL-BASED USER INTERFACE DEVELOPMENT ENVIRONMENT: A SURVEY

To highlight the importance and need of declarative models in UI generation, we present some major processes of user interface generation based on these models. These processes differ from each other in terms of input information, mostly represented by declarative models and from which the generation is effectively done, the generation target, the generation process itself and the tools.

Most MB-UIDEs (Model-Based User Interface Development Environments) use the task and the domain model to specify the user interfaces such as Tritdent, Teallch, Goliath, Fuse such as shown in Table 1. If a model is used, it is marked with "+"; if not with "—"; "+/—" indicates that there is no concrete validation even if the tool is said to support the model.

MB-UIDE	Models				
	Task	Domain	Presenta-tion	User	Dialogue
Trident	+	+	+	—	—
Teallach	+	+	+	+/—	—
Tadeus	+	+	—	+/—	+
Fuse	+	+	—	+/—	+
Goliath	+	+	+	—	—
Janus	—	+	—	—	—
Mecano	—	+	—	—	—
Genius	—	+	—	—	+

Table 1. the different approaches in MB-UIDE and the supported declarative models.

Trident [7] uses task, domain and presentation models to specify the interface. The task model is represented with an Activity Chaining Graph; the domain model by an entity-relationship diagram and the presentation model by the abstract user interface objects.

Teallach [2] uses the task, domain and presentation models to generate the user interfaces. Each model in Teallach defines a view of the information required to generate the interface to a particular application. The domain model describes the underlying application in terns its data and operation; the task model describes what the user can do with the user interface in terms of its dynamic and information processing requirement; and the presentation model indicates how the resulting interface will appear.

In Tadeus [17], like most of other MB-UIDEs, the UI developer has to create the task model which is represented by a hierarchical structure of the tasks; the domain model which presents the important entities of the application; the user model which presents the characteristics of the users. Specially, the dialogue model is semi-automatically created based on three created models. The dialogue model is used to describe the interface in term of views and dialogues.

In FUSE [15], the use of declarative models is similar. The task model is used to describe the task hierarchy of the application model. The domain model is used to describe the functions and the data structure of the relevant user interfaces. The user model is used to describe the static and dynamic properties of the user groups and individual users which influences both the UI generation process and the kind and depth of the help offered by the user guidance component. The dialogue model of Fuse is generated based on three models including task, domain and user models. This model is used to describe the transformation of the task, domain and user models.

Goliath [6] uses the declarative models including the application, presentation and the dialogue models to generate the user interfaces. Goliath's application model is used to define the data types and function signatures. The presentation model is used to describe the basic presentation elements that play a part of the interface. The dialogue model is defined in terms of abstract containers.

Janus [18] emphasizes the use of object-oriented domain model to generate the interface. In this model, a domain object is represented by a class and a class is described by its attributes and methods. During the automatic generation process, a window is generated for each class; its attributes are transformed into controls; and its methods into buttons or menu items.

Like Janus, Mecano [9] uses the domain model to generate the static layout and the dynamic behavior of an interface. The domain model of Mecano is a representation of the objects in a domain and their relationships. The interface model contains all the facets of an interface design.

Genius [8] generates the interfaces for database oriented applications through an existing domain model represented as an extended entity relationship model. In order to structure the information with respect to tasks of the user, the views are defined in the domain model. A view consists of a subset of entities, relationships and attributes of the overall data model. The dialogue model is generated from these views.

Design tools, such as those proposed in Table 2, are essential for modeling the tasks, domain's objects, UI's objects … and for generating the interfaces.

MB-UIDE	Tool	Language used for describing generated user interface
Trident	SEGUIA SIERRA	AION/DS User interface language
Teallach	Teallach	Java code
Tadeus	Tadeus	Textual description for

		UIMS
Fuse	FIRE / FLUID BOSS PLUG-IN	C++ code
Goliath	Goliath's design tool	Caml
Janus	OOA-Tool	C++ code
Mecano	Browser tool Intelligent designer tool	
Genius	ER diagram editor ER : Entity Relationship	Textual description

Table 2. The different approaches in MB-UIDE and their tools.

SEGUIA and SIERRA tools developed for the Trident [7] approach. SEGUIA tool semi-automatic generates the user interface by making questions and suggesting responses to the designer. SIERRA tool generates the guidelines for designing user interface.

Teallach [19] provides three separate editors for manipulating the models. Most of the modeling operations are performed ergonomically by drag-and-drop or cut-and-paste. The user interface is generated in java.

In Tadeus [17], the user interface prototype is automatically generated from the task, domain and dialog models. This prototype is described in terms of textual descriptions which can be implemented throught the tool User Interface Management System.

FUSE [15] consists of the BOSS (BedienOberflächen-SpezifikationsSystem), FLUID (FormaL User Interface Development), PLUG–IN (PLan–based User Guidance for Intelligent Navigation) and FIRE (Formal Interface Requirements Engineering) components that may also be used independently. FIRE provides graphical editors for setting up the task, domain and user models. FLUID is used to generate the specification of static and dynamic properties of a logical user interface. Based on this logical user interface, BOSS is used to generate the implementation of the user interface in terms of C++ code. Finally, PLUG-IN is used to generate the user guidance components.

Goliath [6] provides a graphical editor for modeling the task, domain and presentation models. The interface generated by Goliath is a complete one implemented in Caml.

OOA-Tool (***Object-Oriented Analysis***) in Janus [18] generates user interfaces described in C++ code and designed to be as ergonomic as possible.

Mecano [9] provides two separate tools: a browser and a intelligent designer. The browser tool is used to define, review and inspect the Mecano's domain model. Based on this model, dialog specifications are generated. These specifications are classified into high-level and low-level dialogs. The Intelligent tool creates the abstract interface object based on the created dialogs.

In Genius [8], models are created by an ER diagram editor to generate the executable user interfaces for database-oriented applications. The generated user interface is described by a specific User Interface Management System, but there is no user interface editor.

Most approaches discussed above try to generate the complete and executable user interface based on some of the task, domain, user, presentation, and dialog models. Specially, these approaches have used the domain model although this model is differently defined in the different approaches. Beside of user interface generation, some of the approaches have also generated the application code for performing the generic application functions such Teallach, Goliath. Most of these functions have a connection with the edition of data in the database. This also is a goal of DB-USE, but unlike with the current approaches, these functions will be automatic created by the DB-USE system instead of being created by the designer. Teallach specifically allows create the interface to non database application by creating domain components which don't derive from the underlying database but derive from the library of Java [19]. This is an important point that DB-USE doesn't use yet.

DB-USE MODELS

We describe in this section the main declarative models used in DB-USE.

Task Model

The task model records the tasks potential end-users of the system may need to perform to do their jobs, independently of dealing with a particular computer platform [20]. Many designs of an interactive system are generated to support these tasks. The task model provides support for modeling both the structure of the tasks and the flow of information between the tasks when carrying out the tasks. The structure of the tasks in the task model is described by the relationships between these tasks. These relationships can be of various types such as temporal and semantic relationships.

DB-USE uses the task model to 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 [13].

DB-USE's task model is transformed into UIs starting with the ConcurTaskTrees Environment-CTTE [20]. User interface generation will then be ensured by DB-USE .

The task model described by CTTE is created at the analyst level. In order to reuse it in our process, it has to be transformed into a task model at the design level. At this level, the task model has to describe the function the user can do with an interface, how the user can interact with the system. DB-USE's task model is used to derive the interface. Its task types differ from CTTE and are defined as follows:

- An Action task describes 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.
- An Operation task describes the display of information to the end-user or the reception of the information from the end-user.

Task mapping	
Abstraction task	Action task
Interaction task (Control type)	Action task
Interaction task (Edit\Monitoring\Selection)	Operation task
Cooperation task	Action task
Application task	Operation task
User task	None

Table 3. Task Mapping.

Table 3 illustrates the mapping of abstract, interaction, application and user tasks from the CTTE's task model at the analyst level to action and operation tasks of DB-USE's task model at the design level. Figure 1 depicts an example of the mapping between tasks in ConcurTaskTrees and in our process.

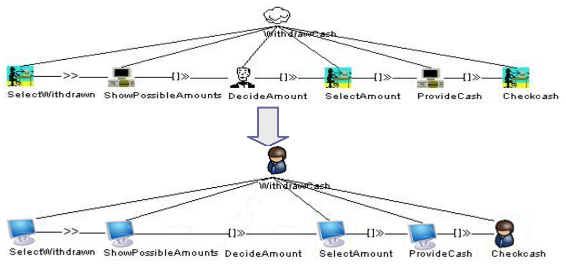
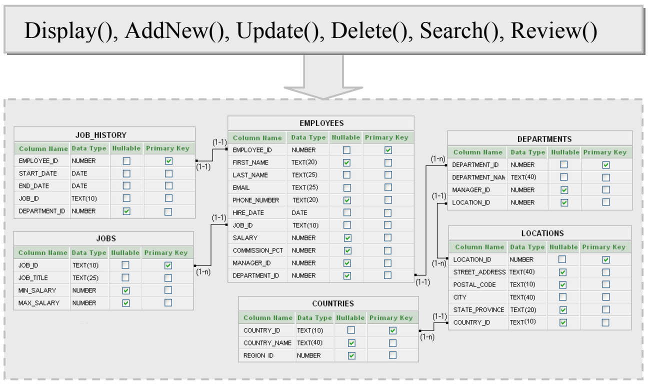


Figure 1. Mapping of *WithDrawCash* task from the task described in CTT to the task described in DB-USE.

In the new task model, the structure, the relationship between the supertask and its subtasks, the temporal operators are all reused for the syntactic and semantic aspects.

Domain Model

A domain 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. A domain model may thus include a data model of the domain. In our process, the domain model will be built from a certain database (See Figure 5).



JOB_HISTORY

Column Name	Data Type	Nullable	Primary Key
EMPLOYEE_ID	NUMBER	☐	✓
START_DATE	DATE	☐	☐
END_DATE	DATE	☐	☐
JOB_ID	TEXT(10)	☐	☐
DEPARTMENT_ID	NUMBER	✓	☐

JOBS

Column Name	Data Type	Nullable	Primary Key
JOB_ID	TEXT(10)	☐	✓
JOB_TITLE	TEXT(25)	☐	☐
MIN_SALARY	NUMBER	✓	☐
MAX_SALARY	NUMBER	✓	☐

EMPLOYEES

Column Name	Data Type	Nullable	Primary Key
EMPLOYEE_ID	NUMBER	☐	✓
FIRST_NAME	TEXT(20)	✓	☐
LAST_NAME	TEXT(25)	☐	☐
EMAIL	TEXT(25)	☐	☐
PHONE_NUMBER	TEXT(20)	✓	☐
HIRE_DATE	DATE	☐	☐
JOB_ID	TEXT(10)	☐	☐
SALARY	NUMBER	✓	☐
COMMISSION_PCT	NUMBER	✓	☐
MANAGER_ID	NUMBER	✓	☐
DEPARTMENT_ID	NUMBER	✓	☐

COUNTRIES

Column Name	Data Type	Nullable	Primary Key
COUNTRY_ID	TEXT(10)	☐	✓
COUNTRY_NAME	TEXT(40)	✓	☐
REGION ID	NUMBER	✓	☐

DEPARTMENTS

Column Name	Data Type	Nullable	Primary Key
DEPARTMENT_ID	NUMBER	☐	✓
DEPARTMENT_NAM	TEXT(40)	☐	☐
MANAGER_ID	NUMBER	✓	☐
LOCATION_ID	NUMBER	✓	☐

LOCATIONS

Column Name	Data Type	Nullable	Primary Key
LOCATION_ID	NUMBER	☐	✓
STREET_ADDRESS	TEXT(40)	✓	☐
POSTAL_CODE	TEXT(10)	✓	☐
CITY	TEXT(40)	☐	☐
STATE_PROVINCE	TEXT(20)	✓	☐
COUNTRY_ID	TEXT(10)	✓	☐

Figure 2. DB-USE's domain model – Employee management.

It is used to identify the attributes of a concrete user interface and to build the important functions of a database application. Besides, in order to obtain the desired behavior of a database application task, the generic functions are also defined in the model; these functions are, for instance, Display(), AddNew(), Update(), Delete(), Search(), Review(), Cancel() and Exit() functions. Unlike Teallach and Goliath, the generic application functions are simply defined in the domain model and they will be analyzed and built in more detail by the system once the function is linked to a concrete task in a concrete context.

User Model

The user model describes the characteristics of the desired users or groups of users such as experience, skill, knowledge, character. The main purpose of a user model is to support the user interface designer to create the user interfaces which tend to the preference of the user. The user model captures capabilities and limitations of the user population, for instance the kind of interaction techniques that are available for visually disabled people differs from the techniques for other human beings. The user model plays an important role in the user interface design; based on it, the designer will specify a complex or a simple user interface. For instance, creating the user interface for the experimented users who have experience with the software applications is different from creating the user interface for newcomers who have never used the computer.

User characteristics can be classified as application-independent or -dependent. Application independent characteristics include preferences, capabilities, psychomotor skills, etc. Application dependent characteristics include goals, knowledge of system and application, etc. In the DB-USE's user model, the users of the system are grouped into three groups (Complex, Mean and Simple) based on the user characteristics mentioned above. Figure 3 depicts DB-USE's user model which describes the user's experience, cultural and psychological characteristics. Then users are grouped based on these characteristics.

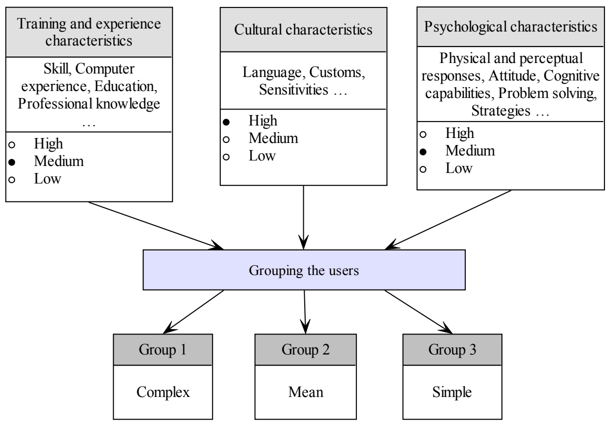

Training and experience characteristics
Skill, Computer experience, Education, Professional knowledge …
High
Medium
Low
Cultural characteristics
Language, Customs, Sensitivities …
High
Medium
Low
Psychological characteristics
Physical and perceptual responses, Attitude, Cognitive capabilities, Problem solving, Strategies …
High
Medium
Low
Grouping the users
Group 1
Complex
Group 2
Mean
Group 3
Simple

Figure 3. Grouping users based on characteristics.

ENGINEERING USER INTERFACE FROM COMBINING TASK, DOMAIN AND USER MODELS

Figure 4 depicts the main components and the architecture of DB-USE. The Model editor agent uses the task-, database- knowledge bases to load the task model from a XML file, the domain model from a concrete database and the user model from a diagram file. The loaded tasks have already been manually linked to the attributes of the domain objects. From these linked objects, the UI builder agent automatically creates the user interface (UI) objects based on the mapping rules [1]. The Function editor agent uses the Function description base to build the functions of the application which are defined in domain model. Once the UI objects have been created and the functions structured, the code generator agent produces the code. Specifically, our process does not only generate the user interface code, but also the application code behind needed to perform these pre-determined functions. In summary, the components of DB-USE are:

- The Task-knowledge base that describes the rules of the task model.
- The Mapping rules base that describes the rules for specifying the concrete user interface from domain's objects and the relationships between these objects and for transforming the concrete user interface to the final user interface [1].
- The Database-knowledge base that describes generic aspects of the database tasks, the advantages of the syntax and the structure of a query.
- The Layout-knowledge base that contains the syntactic design guidelines for controls, windows and other widgets layouts. It also describes the semantic rules from which the control types are defined.
- The Messages base that contains the generic messages such as errors, warnings, information to users messages.
- The Function description base that describes the generic functions of a task of database application.

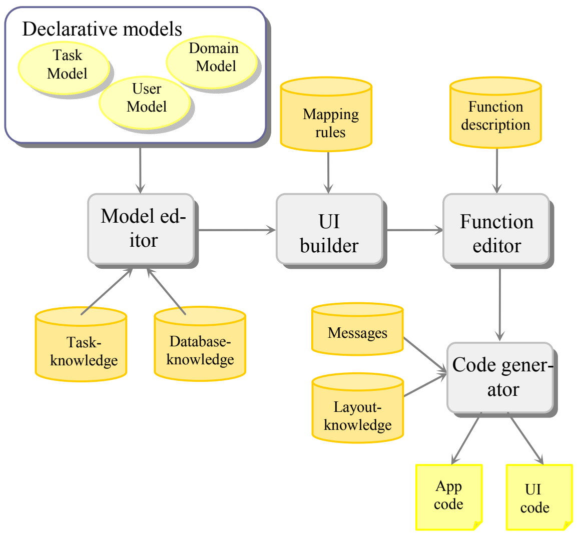


Figure 4. Main components and architecture of DB-USE.

DB-USE 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 (See Figure 5).

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 **Model Editor** is used to load the task model from a XML file, the user model from a diagram file, the domain model from a concrete database. The generic functions of a database application which are defined by DB-USE are included into its domain model. Once these models have been loaded, they are verified and modified by the **Model analyst**. He can change the attributes of the tasks such as name, type or delete an existing task or add a new one. Specially, he can create a new DB-USE task model with the **Model Editor** if it is not created yet.

The associations between the components of the task and domain models are made by the **UI Designer** in the relation making phase. Firstly, **UI Designer** has to specify a composite action task that he wants to generate the UI. Then DB-USE displays its sub-tasks based on rules RST (**R**ules for selecting the **S**ub-**T**asks). The operation tasks are linked to the domain's attributes and the action tasks are linked to the functions defined in domain model (See Figure 6). One operation task can be linked to more than one domain's attribute; one domain's attribute can be linked to more than one operation task. But one action task can be linked to only one function.

- **RST1:** All sub-tasks at level 1 are selected.
- **RST2:** All sub-tasks of a sub-task of task type operation is selected.

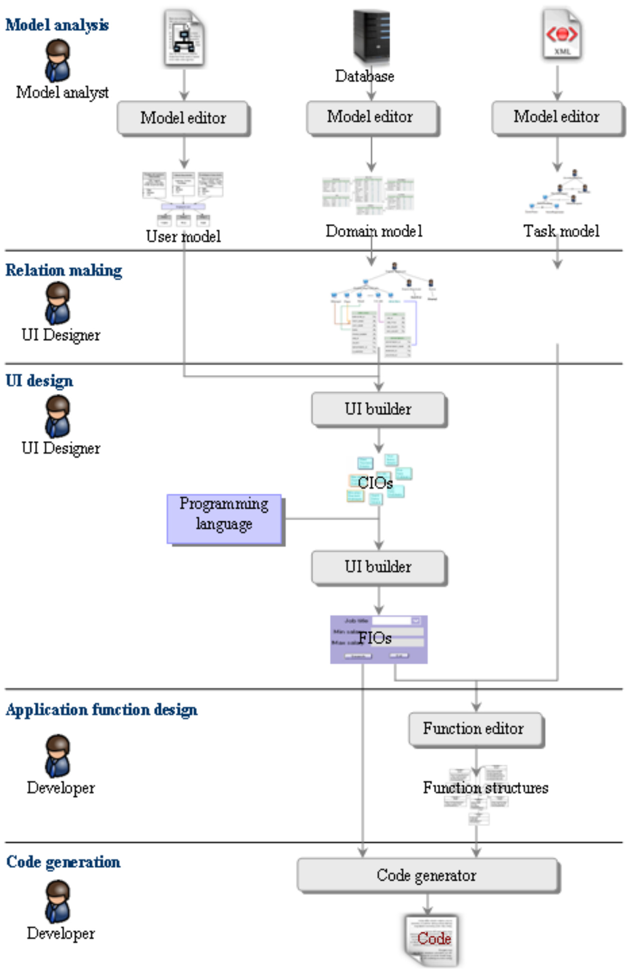


Figure 5. The metrological process of DB-USE.

In the UI design phase of the DB-USE process, the CIOs [16] (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 CIOs have been created, in order to reuse them in the different running environments (including platform, device and language); DB-USE stores them in terms of UsiXML (USer Interface eXtensible Markup Language) [23]. Since the programming language has been determined, the FIOs (Final Interaction Object) are created based on the CIOs (See Figure 7).

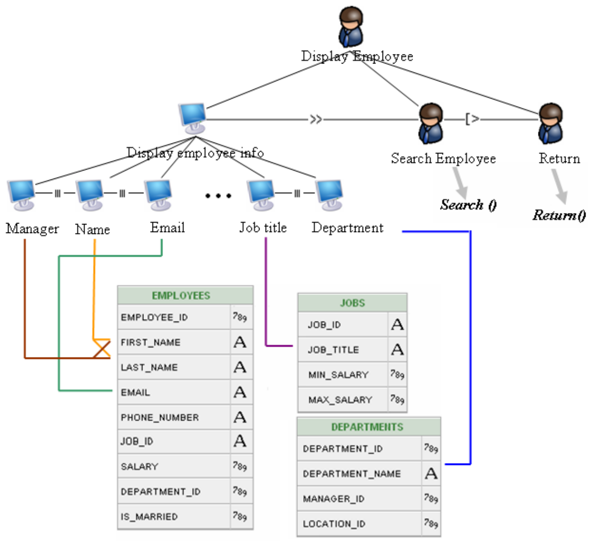


Figure 6. Associations between tasks and domain's components.

In order to specify the attributes of the UI objects, DB-USE uses the **Mapping rules** [1] 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 a crucial role since he has to make sure that the created UI objects satisfy the goal of the tasks.

The application function design phase starts once the FIOs have been completely created. The **Function** base 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**.

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 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.

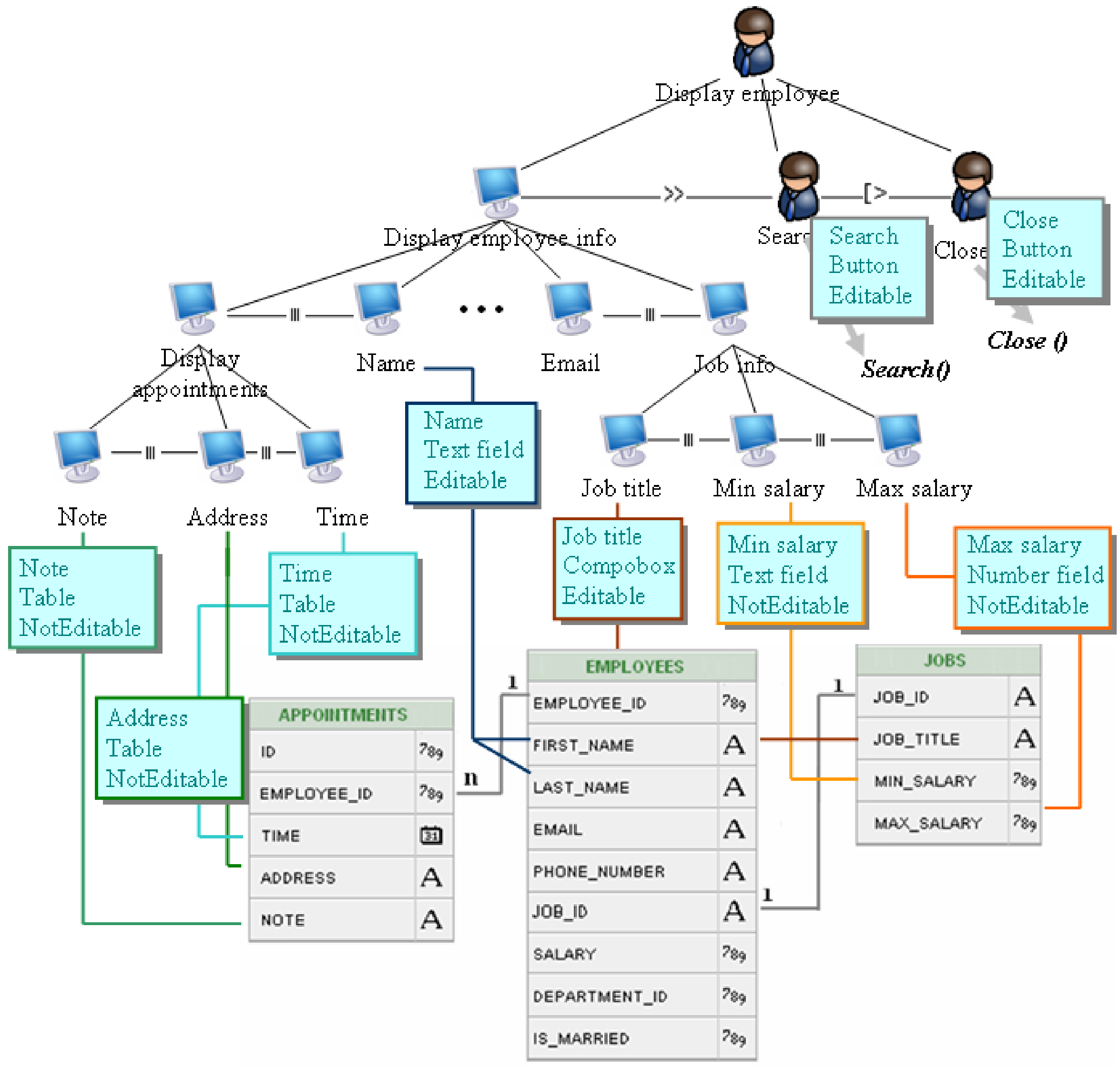


Figure 7. Creating FIOs based on the CIOs.

DB-USE TOOL

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

A presentation

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 the functions defined in the domain model. These important 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 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 8 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

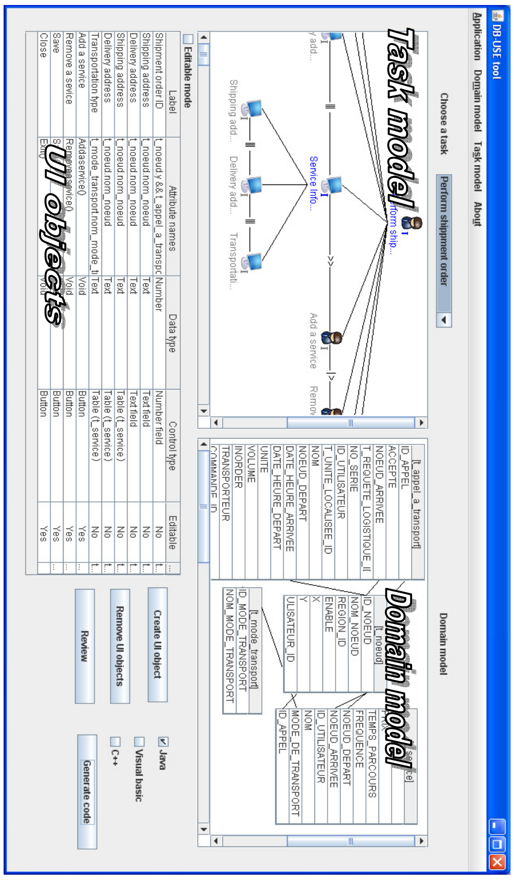


Figure 8. DB-USE Tool.

A concrete example - *Edit Employee Information*

In order to illustrate how the user interface code and application code are generated by DB-USE, a simple example is showed. The goal of this example is to allow the end-user review the employee information, create a new employee, delete an exiting one, and modify the employee information.

DB-USE performs this example through six steps.

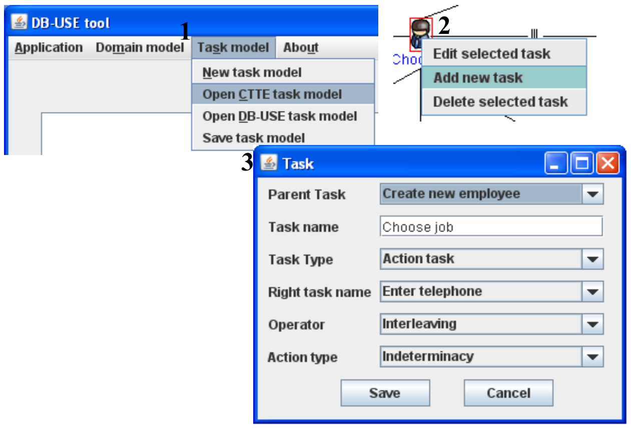
Step 1: *Loading* ***Edit Employee Information*** *Task model*

The task model may be loaded from an existing CTTE task model or created by using DB-USE. (See Figure 9)

(1) in Figure 9 allows the designer to create a new task model, load an existing one, or save it in term of DB-USE's task model.

(2) in Figure 9 allows the designer to edit the tasks in the task model.

(3) in Figure 9 is the task dialogue which is used to modify the task's information.



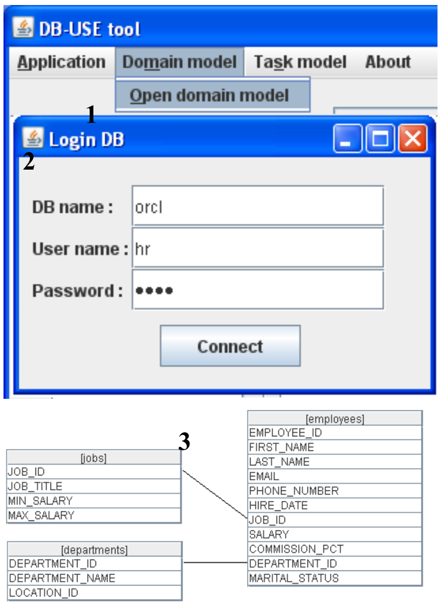
Parent Task	Create new employee
Task name	Choose job
Task Type	Action task
Right task name	Enter telephone
Operator	Interleaving
Action type	Indeterminacy

Figure 9. Loading or creating task model.

Edit Employee Information task model is showed in Figure 10.

Step 2: *Loading domain model*

The domain model is loaded from Oracle database.



DB name :	orcl
User name :	hr
Password :	••••

[jobs]
JOB_ID
JOB_TITLE
MIN_SALARY
MAX_SALARY

[departments]
DEPARTMENT_ID
DEPARTMENT_NAME
LOCATION_ID

[employees]
EMPLOYEE_ID
FIRST_NAME
LAST_NAME
EMAIL
PHONE_NUMBER
HIRE_DATE
JOB_ID
SALARY
COMMISSION_PCT
DEPARTMENT_ID
MARITAL_STATUS

Figure 11. Loading domain model.

(1), (2) and (3) are the steps performed to load the domain model. *Edit Employee Information* task involves to only three objects including *Jobs*, *Departments*, and *Employees*. So the other objects are not showed in this case. Beside of creation of objects, the DB-USE also creates the application functions in this model. These functions are showed in the function popup menu of Figure 11.

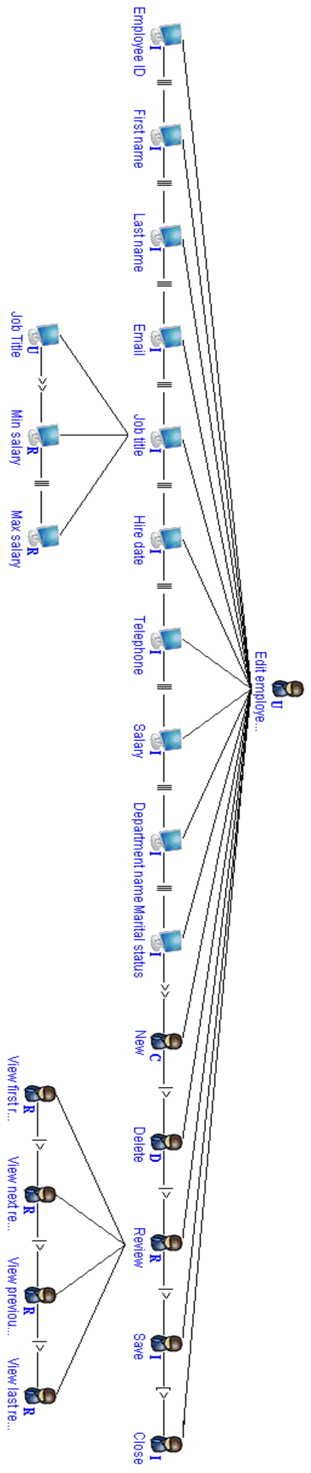


Figure 10. ***Edit Employee*** **task model.**

Step 3: *Making the associations between the tasks and the domain's components:*

The association between the operation tasks and the attributes of the domain's objects is created by selecting a concrete task and the attributes then just simply clicking on the "Create UI object" button. The association between the action tasks and defined application functions is created by selecting a concrete task, with the mouse, then selecting the application function in the menu (See Figure 11).

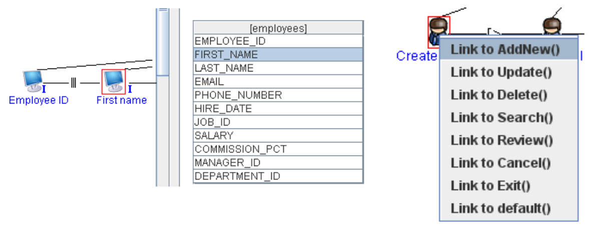
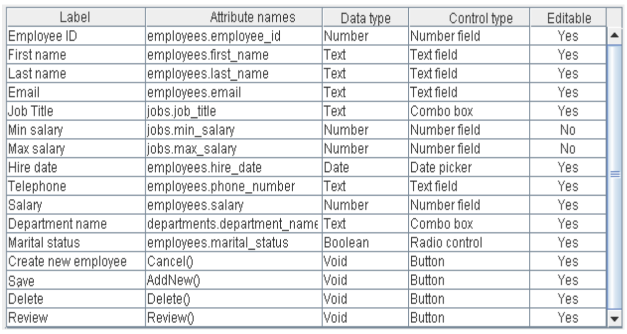


Figure 11. Making the associations between the tasks and the domain's attributes, the generic functions.

For *Edit Employee Information* task: The *Employee id*, *First name*, *Last name*, *Email*, *Hire date*, *Telephone*, *Salary*, *Marital* status tasks are liked to *Employe_ID*, *First_Name*, *Last_Name*, *Email*, *Hire_Date*, *Telephone_Number*, *Salary*, *Marital_Status* attributes of the *Employees* object. The *Create new employee* task is liked to cancel() function to reset the control to empty; the *Save* task is linked to AddNew() function to insert to data to the database; the *Delete* task is linked to Delete() function to delete the current record; the *Review* task is linked to Review() function to review the first, previous, next and last records; the *Close* task is linked to Exit() function to close the form.

Step 4: *Creating the UI objects.*



Label	Attribute names	Data type	Control type	Editable
Employee ID	employees.employee_id	Number	Number field	Yes
First name	employees.first_name	Text	Text field	Yes
Last name	employees.last_name	Text	Text field	Yes
Email	employees.email	Text	Text field	Yes
Job Title	jobs.job_title	Text	Combo box	Yes
Min salary	jobs.min_salary	Number	Number field	No
Max salary	jobs.max_salary	Number	Number field	No
Hire date	employees.hire_date	Date	Date picker	Yes
Telephone	employees.phone_number	Text	Text field	Yes
Salary	employees.salary	Number	Number field	Yes
Department name	departments.department_name	Text	Combo box	Yes
Marital status	employees.marital_status	Boolean	Radio control	Yes
Create new employee	Cancel()	Void	Button	Yes
Save	AddNew()	Void	Button	Yes
Delete	Delete()	Void	Button	Yes
Review	Review()	Void	Button	Yes

Figure 12. User interface object are displayed in terms of text.

Once an association has been created, a UI object is automatically created by DB-USE. The information of this UI object is displayed in a table as show in Figure 12 This information includes the label of the control, the attribute linked, the data type of these attributes, the control type and the editable attribute of control.

Step 5: *Storing the UI objects by using UsiXML*

The created UI objects will be described in UsiXML such as represented in Figure 13 by selecting ***Application \ Save UI as UsiXML*** menu. The objective of using the UsiXML is that the UI can be implemented in the different environments. Figure 13 depicts how to save the UI objects in terms of UsiXML.

```
<outputText id="id" name="Employee ID lable" content="xpath" defaultContent="Employee ID" isVisible="true"
                                                              isBold="false" textColor="#000000"/>
<inputText id="id0" name="Employee ID control" maxLength="50" numberOfColumns="1" isEditable="true"/>
<outputText id="id1" name="First name lable" content="xpath" defaultContent="First name" isVisible="true"
                                                             isBold="false" textColor="#000000"/>
<inputText id="id2" name="First name control" maxLength="50" numberOfColumns="1" isEditable="true"/>
<outputText id="id3" name="Last name lable" content="xpath" defaultContent="Last name" isVisible="true"
                                                            isBold="false" textColor="#000000"/>
<inputText id="id4" name="Last name control" maxLength="50" numberOfColumns="1" isEditable="true"/>
<outputText id="id5" name="Job title lable" content="xpath" defaultContent="Job title" isVisible="true"
                                                        isBold="false" textColor="#000000"/>
<comboBox id="id6" name="Job title control" isEnabled="true" currentValue=""/>
<outputText id="id7" name="Department name lable" content="xpath" defaultContent="Department name"
                                               isVisible="true" isBold="false" textColor="#000000"/>
<comboBox id="id8" name="Department name control" isEnabled="true" currentValue=""/>
<button id="id9" name="Add a new employee" content="xpath" defaultContent="New"/>
```

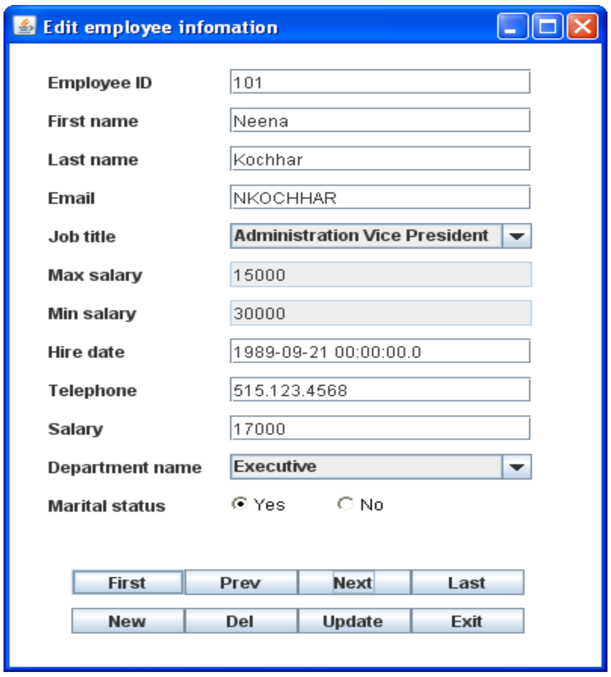
Figure 13. Some UI objects in Figure 12 are described by using UsiXML.

Step 6: *Creating the final user interface*

Based on the programming language determined by the designer and the created concrete interaction objects, the final interaction objects are automatically specified.

In order to help the designer to materialize the user interface that will be generated during the last step of the process, DB-USE has to allow to review the user interface in terms of graphics.

DB-USE generates the user interface code based on the user interface objects that have been created and the application code based on the selected functions. The code generator produces java code by using graphical libriaries such as Swing and AWT. Unlike the current applications, DB-USE generates both the user interface and application codes. The generated code is complete and can be compiled and ran immediately when usually code generated by other applications are just prototypes or code skeletons.



Employee ID	101
First name	Neena
Last name	Kochhar
Email	NKOCHHAR
Job title	Administration Vice President
Max salary	15000
Min salary	30000
Hire date	1989-09-21 00:00:00.0
Telephone	515.123.4568
Salary	17000
Department name	Executive
Marital status	[✓ Yes] No

Figure 14. ***Edit Employee Information*** **form.**

Figure 14 depicts *Edit Employee Information* form that is generated from the previous steps.

CONCLUSION

To be efficient, data-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 application code for performing the generic functions of the database applications. In future, the navigation between the dialogues will be automatically specified based on the operators of the task; the position and dimension attributes of UI object will be optimized; and finally, allow define the data type which do not come from the database.

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. Tran, V., Vanderdonckt, J., Faulkner, S., Generating User Interface from Task, User and Domain Models. In *Proc. of 2nd Int. Conf. on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services CENTRIC'2009* (Porto, September 20-25, 2009), IEEE Computer Society Press, Los Alamitos, 2009, pp. 19-26.
2. 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.
3. Eisenstein, J., Puerta, A., Adaptation in automated user-interface design. In *Proceedings of the 5th international conference on Intelligent user interfaces*, p.74-81, January 09-12, 2000, New Orleans, Louisiana, United States.
4. Nichols, J., Faulring, A.. "Automatic Interface Generation and Future User Interface Tools". In: *CHI. Proceedings of the Workshop on the Future of User Interface Design Tools*, 2005.
5. Pinheiro da Silva, P., Griffiths, T., N. Paton, 2000. Generating user interface code in a model based user interface development environment. In: *Proceedings of Advanced Visual Interfaces*. ACM Press, New York, pp. 155–160.

6. Julien, D., Ziane, M., and Guessoum, Z.. GOLIATH: An extensible model-based environment to develop user interfaces. In *Proceedings of the Fourth International Conference on Computer Aided Design for User Interfaces*, P. 95–106. Kluwer Academics Publishers, 2004.

7. 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. In *Proc. of 2nd Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95* (Toulouse, 7-9 June 1995), Ph. Palanque, R. Bastide (eds.), Springer-Verlag, Vienna, 1995, pp. 262-278.

8. C. Janssen, A. Weisbecker, J. Ziegler: Generating User Interfaces from Data Models and Dialogue Net Specifications. In *Ashlund S., Mullet K., Henderson A., Hollnagel E., White T. (eds.)*. New York: ACM Press 1993 (pp. 418-423).

9. Puerta, A., "The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development," *Proc. CADUI 96*, J. Vanderdonckt, ed. http://www.info.fundp.ac.be/~jvd/dsvis/cadui96.html.

10. Moroney, L. and MacDonald, M. , ASP.NET Applications in Pro ASP.NET 1.1 in VB .NET From Professional to Expert, Apress, 2006, pp. 183- 230.

11. Myers, B., Hudson, S. E., Pausch, R.. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction* (TOCHI), v.7 n.1, p.3-28, March 2000.

12. 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.

13. 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, P. 77-94.

14. 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 2005. (P. 26-27)

15. Lonczewski, F., Schreiber, S.: The FUSE-System: An Integrated User Interface Design Environment. In: *J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces*. Namur University Press, 1996, 37-56.

16. Vanderdonckt, J. and Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In *Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93* (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 424-429.

17. Elwert, T. and Schlungbaum, E., Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In *Designing, Specification and Verication of Interactive Systems*, pages 193{208, Vienna, 1995. Springer.

18. Balzert, H., From OOA to GUI - The JANUS-System, in *Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction*, Lillehammer, June 1995, pp. 319-324.

19. Barclay, P.J., Griffiths, T., McKirdy, J., Paton, N.W., Cooper ,R., and Kennedy, J.. The Teallach Tool: Using Models for Flexible User Interface Design. In *pro. 3rd International Conference on Computer-Aided Design of User Interfaces*, Louvain-la-Neuve (Belgium), 21-23 October 1999.

20. Paternò, F., Mancini, C., and Meniconi, S., Concurtasktrees: A diagrammatic notation for specifying task models. In *Human-Computer Interaction*, pages 362–369. Chapman and Hall, 1997

21. Pribeanu, C., Tool Support for Handling Mapping Rules from Domain to Task Models. *Coninx, K., Luyten, K., Schneider, K. (Eds.): Proc. of TAMODIA 2006, Hasselt, Belgium*, 23 – 24 October. Lecture Notes in Computer Science - LNCS 4385, Springer 2007, pp. 16-23.

22. Laudon, Kenneth C., Laudon, Jane P., Management Information Systems. Pearson Education (US), 2009.

23. Limbourg, Q., Vanderdonckt, J., Multi-Path Transformational Development of User Interfaces with Graph Transformations, in Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), "Human-Centered Software Engineering", Chapter 6, HCI Series, Springer, London, 2009, pp. 109-140.

24. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest, 2008, pp. 1–10.