A DEVELOPMENT PROCESS OF THE USER-INTERFACES FOR THE DATABASE APPLICATIONS

Adrian ALEXANDRESCU[1]

Proiectarea, implementarea şi testarea interfeţelor-utilizator, din cadrul aplicaţiilor cu baze de date, sunt sarcini ce necesită o muncă semnificativă care, de cele mai multe ori, este o muncă de rutină. Lucrarea propune un proces de dezvoltare bazat pe şabloane de proiectare şi implementare pentru acestea. Procesul de dezvoltare propus facilitează generarea automată a interfeţelor-utilizator plecând de la modelul conceptual al domeniului aplicaţiei şi specificaţiile componentelor interfeţelor-utilizator. Autorul a implementat procesul propus utilizând platforma Microsoft Visual Studio 2008.

The design, implementation and test of the user-interfaces within the database applications, are tasks that require a lot of work, which in many cases is a routine work. The paper proposes a development process based on design and implementation patterns for them. The proposed development process facilitates the automated generation of the user-interfaces starting from the conceptual model of the application domain and the specification of the user-interfaces components. The author has implemented the proposed process using Microsoft Visual Studio 2008.

Keywords: software engineering, user-interfaces development, design patterns for user-interfaces, automated generation of the user-interfaces

1. Introduction

The database applications cover large domains of activity such as: economy, public administration, education, health, defense, etc. Their specificity is that they manage a large number of instances of the concepts from a certain domain. They also manage the relationships between the concepts instances. Usually, the term *entity* is a synonym for *concept instance*. The user-interfaces (UI) of the database applications are software components which offer presentation and update services for entities and their relationships.

User interfaces development is a subject for many conferences and workshops (such as [8], [9]). Model-based interface development, automated and semi-automated techniques for user-interface generation, user-interface design patterns, are frequently discussed issues.

[1] Lecturer, Department of Mathematics and Informatics, Ovidius University of Constanta, Romania, e-mail: aalexandrescu@univ-ovidius.ro

The paper presents a development process for the user-interfaces, based on a set of design and implementation patterns. From the practice of the user-interface development, I have identified some UI design patterns. The use of the patterns opens the way of the UI automated generation. The developer effort will be directed towards the specification of the UI properties, letting the routine work to be done by some predefined or automated generated software.

The section 2 presents a case study of an application domain, modeled by an UML class diagram. The model is then refined in section 3, using the REA modeling. Section 4 shows the database schema for the application domain, and section 5 proposes some patterns which cover the UI design for all the conceptual categories and the their interactions. Section 6 presents some conclusions.

2. An Example of an Application Domain Model

Let consider an application which manages certain activities from a library. Fig. 1 presents a conceptual diagram for the main domain concepts, using an UML class diagram. The *BookStock* is one of the main concepts, whose instances refer the books existing in the library at the current moment. The *Book* concept refers the books owned by the library. *BookTitle* is a concept that defines the titles of the books from the library. For every instance of the *BookTitle* concept, there may be at least one instance of the *Book* concept. *Client* is a concept whose instances represent persons which may borrow books from the library. They also must return the books borrowed. *Borrow* is a concept that has an instance for every borrowing action of a client at a certain moment. A *Borrow* instance has associated one or more instances of the *BorrowDetail* concept, representing the list of the books borrowed. The *Return* concept is similar to the *Borrow* concept. The *Area* concept groups entities from the *BookTitle* concept. Other concepts are *Author* and *Editor*.

There are also presented the relationships between the concepts and their cardinalities.

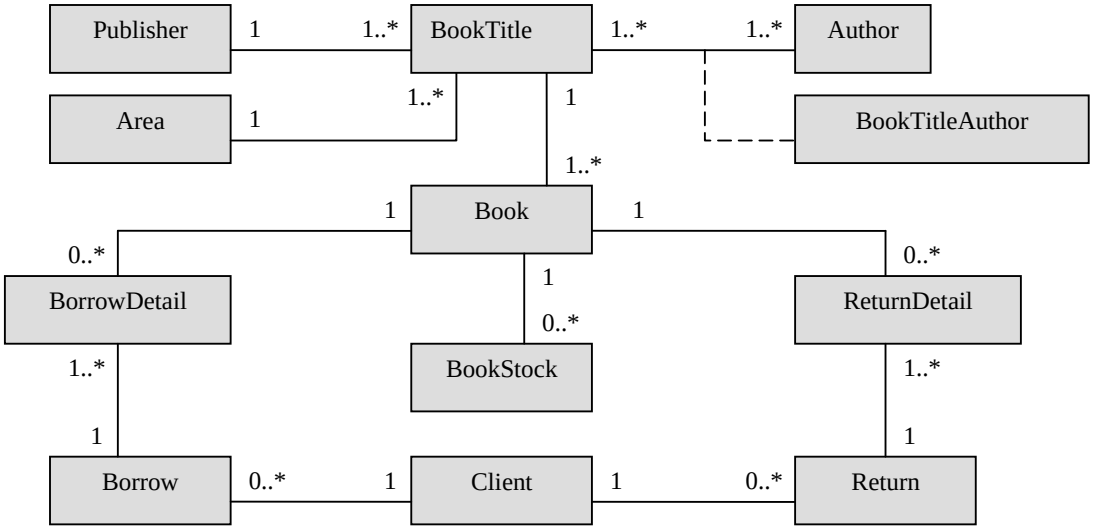

Fig. 1. The conceptual model of the application domain

The *BookTitleAuthor* is a concept that implements the many to many relationship between the concepts *BookTitle* and *Author*.

3. A REA (Resource-Event-Agent) Modeling Perspective

REA was originally proposed as a generalized accounting modeling, by William E. McCarthy in 1982 [1]. Since then, the original REA model has been extended by McCarthy and Guido Geerts to a framework for business systems [2]. Even if the REA modeling is considered suitable only for the economic domain, I consider that it may be applied to other domains too. The example I consider in this paper illustrates this. The REA modeling introduces several conceptual categories. The main categories are: resources, events and agents. A *resource* is something that has an important value and we want to manage. In the example from the Fig. 1, the *BookStock* is a concept from the resource category. An *agent* is a person or an organization that has rights to increase or decrease the value of a resource. In the example from the Fig. 1, the *Client* concept belongs to the agent category. It may decrease the value of the *BookStock* resource by borrowing books and increase the value of the *BookStock* resource by returning books. The

organization itself (the library) may be also considered as an implicit agent. Finally, an *event* is an increase or a decrease of a resource value. In the same example of the Fig. 1, *Borrow* and *Return* are concepts from the event category.

The authors of the REA modeling have extended the three base conceptual categories, presented above, with new ones. Some of them are: *type* and *group*. The *type* conceptual category defines types of instances of a concept. In our example, *Book* is a concept from a *type* conceptual category. It defines types of *BookStock* instances. The *group* conceptual category allows us to group instances from one or more concepts. The *Area* concept is a *group* conceptual category. BookTitle may be considered a *group* concept because it groups the Book instances with the same title. There are many other extensions of the REA modeling, made by its authors.

An extension, done this time by the author of this paper, is the *EventDetail* concept category. A concept from this category describes the content of an event concept. For example, the *BorrowDetail* concept specifies the list of books borrowed by a client at one moment. The *ReturnDetail* is also a concept from the *EventDetail* category. It is optional that an *event* concept be associated with an *EventDetail* concept.

I also consider that there are concepts which can not be included in the categories mentioned above. They may be called simple concepts. The concepts *Author* and *Publisher* may be considered simple concepts. There is also possible to consider Publisher and Author as *group* concepts.

The concept category may label the concepts from the conceptual diagram using stereotypes. The relationships between concepts may be also labeled with predefined names, using the following rules:

- Any event concept must be related to a resource concept, in the case that the event concept has not an associated event-detail concept. If the event concept has an associated event-detail concept, the event-detail concept will be related to the resource concept. The relationship between an event concept (or an event-detail concept) and a resource concept will be labeled *input* if the event concept increases the value of the resource concept, or *output* if the event concept decreases the value of the resource concept.
- Any event concept must be also related to an agent concept. The relationship between an agent concept and an event concept that increases the value of the resource concept will be labeled with the name *supply*, and in case of an event concept that decreases the value of the resource concept, the label will be *receive*.
- The relationship between a concept and a type concept will be labeled with the name *specification* [3].
- The relationship between a concept and a group concept will be labeled with the name *grouping* [3].

- The relationship between an event concept and its associated event-detail concept will be labeled with the name *detail*.
- The *input* and *output* relationships need not the cardinality specification. We'll consider, in the context of the conceptual diagram, that a resource concept represents the evaluation of the resource instances at the current moment.

With these considerations the conceptual diagram from Fig. 1 becomes the REA conceptual diagram from the Fig. 2. The Fig. 2 contains also the attributes for the concepts.

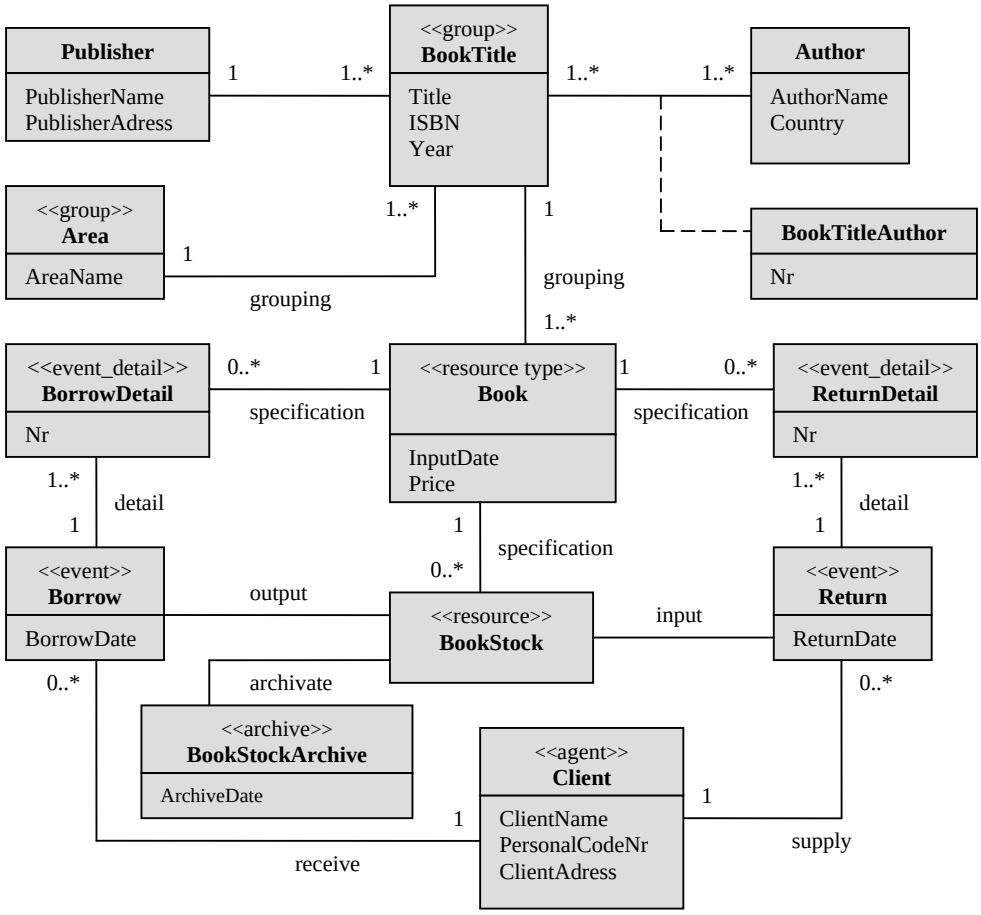

Fig. 2. The REA conceptual diagram for the application domain

In this paper, I consider that the resource concept refers to the current value of a resource. We may also discuss about the initial value of a resource,

when started the activity of the enterprise or organization. We may also consider as initial value of a resource at the moment when the application is implemented. We may also consider the value of a resource at any moment, for example at the beginning of every month. For this reason, every resource concept needs an associated *resource-archive* concept which stores the resource values at previous moments. The resource-archive concept is another extension of the REA modeling, done in this paper, and may be represented in the conceptual diagrams with the stereotype <<*archive*>>. The resource-archive concept has all the attributes of the resource concept and supplementary has an *ArchiveDate* attribute which means the moment of the resource value.

The archive concept may also be used for the event concepts (and its associated event-detail concepts). The reason, in this case, is to reduce the number of records from the database relation associated to the event concept, thus increasing the application performances without limiting the access to the concept instances. When the user decides to archive instances of the event concepts, these are moved from the database relation, associated to the event concept, to the database relation associated to the *event-archive* concept. The event-archive relation schema is the same as the event relation schema.

The REA conceptual categories enable an easier identification of the design and implementation patterns for the user-interfaces corresponding to the domain concepts.

4. The Database Schema Generation

Once the conceptual diagram is established, we may generate the relational database schema. The generation of the database schema may be automated [4], taking into account that the concepts belong to some predefined categories. The automated generation of the database starts from the description of the REA conceptual diagram and the specification of the concepts attributes. For the example of the Fig. 2, we obtain the following database schema:

Publisher (PublisherId, PublisherName, PublisherAdress)
Author (AuthorId, AuthorName, Country)
Area (AreaId, AreaName)
BookTitle (BookTitleId, Title, ISBN, AreaId, PublisherId, Year)
BookTitleAuthor (BookTitleId, Nr, AuthorId)
Book (BookId, BookTitleId, Price, InputDate)
BookStock (BookId)
BookStockArchive (ArchiveDate , BookId)
Client (ClientId, ClientName, PersonalCodeNr, ClientAdress)
Borrow (BorrowId, BorrowDate, ClientId)
BorrowDetail (BorrowId, Nr, BookId)

Return (ReturnId, ReturnDate, ClientId)
ReturnDetail (ReturnId, Nr, BookId)

5. The Patterns for the Conceptual Categories User-Interfaces

One of the main principles of the process presented here is that every concept of the application domain needs its own user-interface (UI). The process relies on a predefined set of UI patterns which cover all the conceptual categories. The UI patterns provide standard functionality for the conceptual categories. The developer may add supplementary functionality to a specific user-interface of a concept. The paper proposes three types of UI patterns which cover the user-interfaces for all the conceptual categories:

UI_G : a general UI pattern for the following conceptual categories: agent, type, group and simple concepts.

UI_E : an UI pattern for the event conceptual category.

UI_R : an UI pattern for the resource conceptual category.

Each type of UI pattern may be specified from three perspectives: structural, functional and behavioral. The structural perspective describes the components of the UI pattern. The functional perspective specifies the standard operations that may be demand by the user for that type of user-interface. The behavioral perspective specifies how the user-interface reacts to the events generated by the user.

The concepts have inherent relationships, as we may see in the conceptual diagram (Fig. 2). Every relationship, between two concepts, may generate an interaction between the corresponding user-interfaces. So we must have also an interaction pattern between two user-interfaces.

5.1. A Proposal for the General User-Interface Pattern UI_G

The structural specification of the UI_G pattern may be done with the UML class diagram from the Fig. 3.

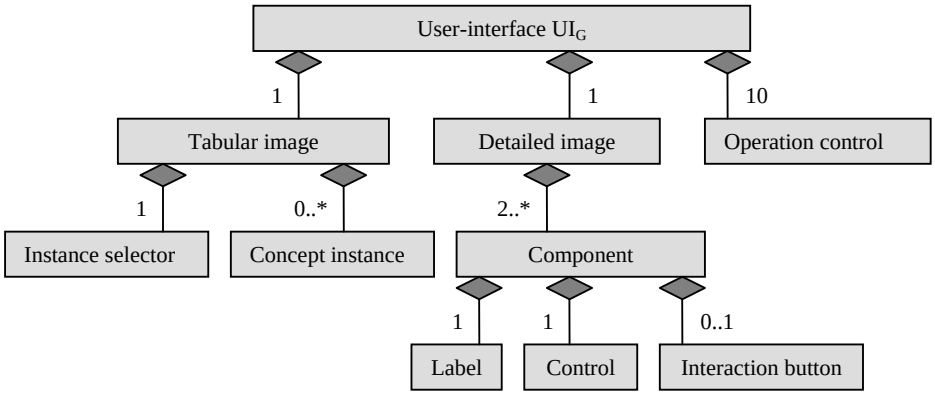

Fig. 3. The structure of the UI_G pattern

The user-interface from the Fig. 3 contains a *tabular image*, a *detailed image* and a set of ten standard *operation controls*. The *tabular image* contains the instances of the concept and an instance selector. The *detailed image* contains at least two components representing fields of certain database relations. A component consists of a label, a control and an optional interaction button. The control displays the value of a field from a database relation and the interaction button initiates the interaction with another user-interface. The ten operation controls starts some standard functions for the user-interface.

The functional specification contains a default visualization function for the concept instances in the tabular image and ten standard functions associated with the operation controls (buttons and checkboxes) of the user-interface:

1. Detailed visualization for the current instance.
2. Select one or more field values to be transferred to a user-interface which has initiated an interaction with the current user-interface.
3. Filter the concept instances of the tabular image.
4. Sort the concept instances of the tabular image.
5. Find a concept instance in the tabular image.
6. Enter the session for adding new concept instances.
7. Modify the current concept instance.
8. Delete the selected concept instances.
9. Save a new concept instance in the database (in the case of an adding session) or save the updates of a concept instance in the database (in the case of modifying a concept instance).
10. Cancel the session of adding new concept instances or cancel the updates of a concept instance.

The users have some rights established by the application administrator, regarding the operations they may use for a certain UI concept and even the UI concepts they may access. It is the responsibility of the application and user-interfaces to determine the user rights from the administrator database, enabling or disabling user access to concepts and operations.

The objects of the UI_G pattern have attributes whose values must be specified when we want to develop a user-interface for a specific concept. For example, the object *tabular image* from the Fig. 3 has an attribute *data-source* which specifies the SQL query string that provides data for the tabular image. The pattern structure description and the attributes values of the pattern objects may be considered as pattern specification. In this paper there are not presented the objects attributes of the UI patterns.

Fig. 4 contains an example of a user-interface implementation, using the UI_G pattern. The associated concept is *BookTitle*. The detailed visualization function is on. Some of the operation controls are not visible because they can not operate in the visualization context. We may observe that the detailed image contains some supplementary information than the tabular image. There is the list of BookTitle authors which is rendered only by the detailed image.

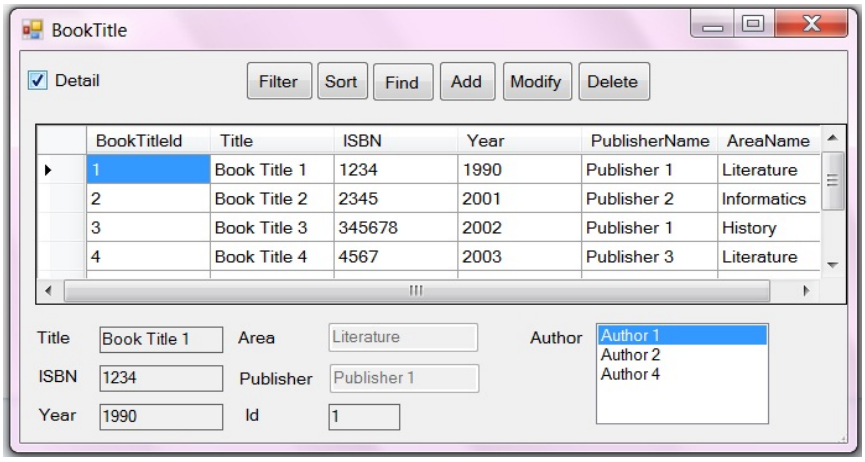

Fig. 4. An implementation of the BookTitle user-interface

We may also consider an image of the book cover that we may present in the detailed image. There is not necessary that the Id field to be visible in the tabular or detailed image.

The behavior of the user-interface may be described by state transition diagrams, interaction diagrams or activity diagrams. For example the scenario for

adding new concept instances may be described with the state transition diagram from Fig. 5.

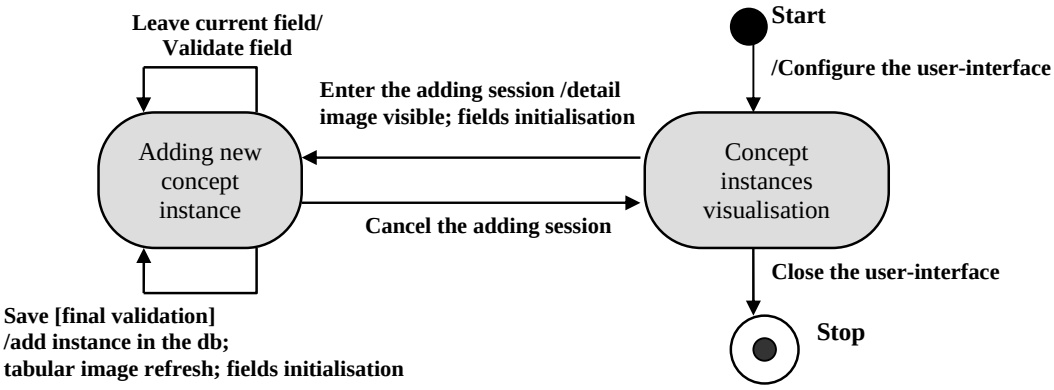

Fig. 5. The behavior for adding new concept instances

The description of the standard functions reveals a set of parameterized sub-functions which may be stored in a library shared by all the user-interfaces. In the case of the functionality described in the Fig. 5, the actions that label the transitions may be considered such sub-functions.

5.2. A Proposal for the Event User-Interface Pattern UI_E

We'll consider that the event concept has an associated event-detail concept. The case of the event concept without an associated event-detail concept may use the UI_G pattern.

The UI_E pattern consists in two different patterns: UI_{E1} and UI_{E2} each of them responsible for some operations. The Fig. 6 presents the structure of the UI_{E1} pattern and Fig. 8 presents the structure of the UI_{E2} pattern.

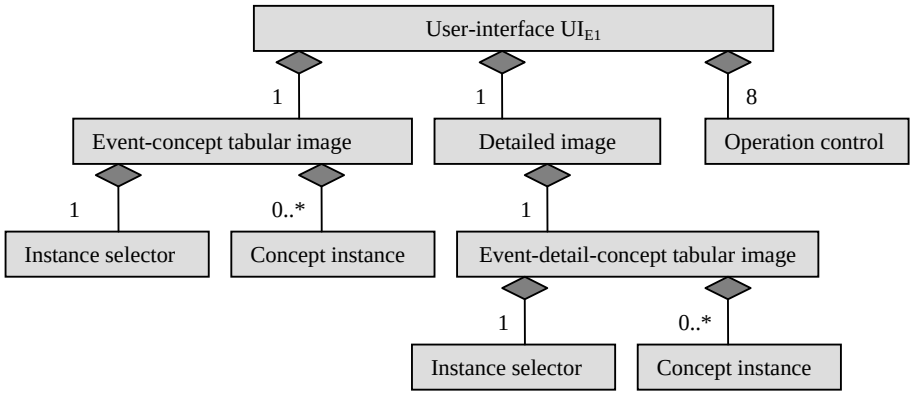

Fig. 6. The structure of the UIE_1 pattern

The user-interface from Fig. 6 contains a *tabular image*, a *detailed image* and a set of eight standard *operation controls*. The *tabular image* contains the instances of the event concept and an instance selector. The *detailed image* contains a tabular image with the instances of the associated event-detail concept and also an instance selector. The eight operation controls implement the standard functions for the user-interface, and they are similar to the first eight functions from UI_G pattern.

Fig. 7 is an example of the implementation of the UI_{E1} pattern for the *Borrow* event concept. The detailed visualization function is on. The upper tabular image contains the instances of the Borrow event concept. The lower tabular image shows the BorrowDetail instances related to the current Borrow instance.

The *Add* operation enables us to insert a new instance of the borrow event concept, opening the user-interface from Fig. 9. In a similar manner acts the *Modify* operation. The *Delete* operation removes from the database the event instance and the related event-detail instances.

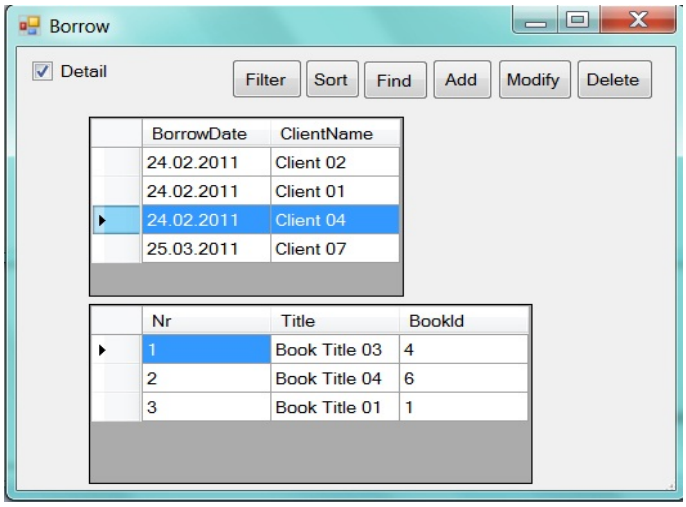

Fig. 7. An implementation of the Borrow UIE_1 user-interface

Fig. 8 presents the structure of the second part UI_{E2} of the user-interface pattern, for the event concepts. UI_{E2} is used when we add a new instance for the event concept (event instance presentation) and the set of related instances of the event-detail concept (event-detail tabular image). An event-detail instance presentation, containing at least two components, is used for the event-detail current instance presentation, in a single record format, and also for updating the event detail tabular image. The five operation controls provide a set of standard functions for the UI_{E2} user-interface: (1) the detailed presentation of the tabular

image current instance, (2) append a new event-detail instance in the tabular image, (3) update the current instance of the tabular image, (4) insert a new instance in a specified position in the tabular image, and (5) delete the current instance of the tabular image.

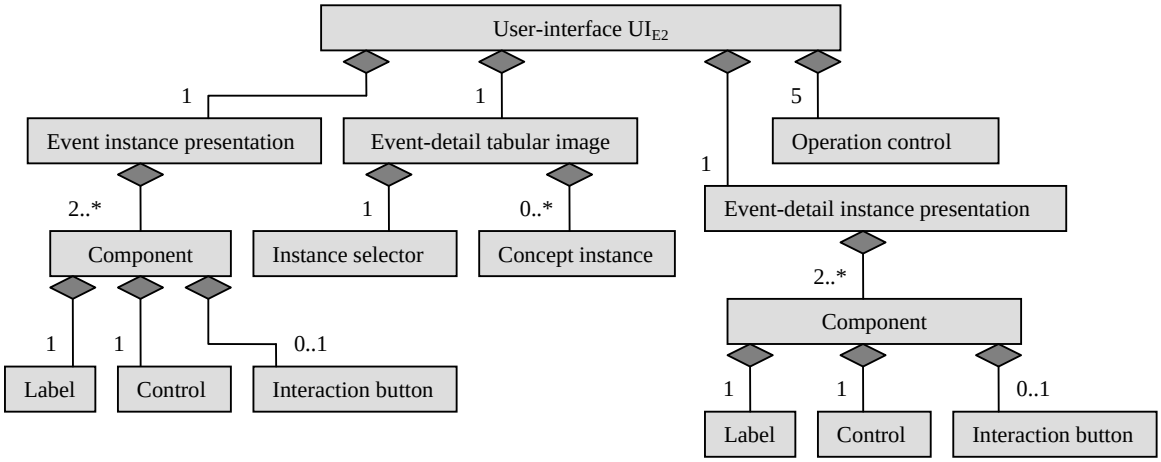

Fig. 8. The structure of the UIE$_2$ pattern

Fig. 9 contains an example of the implementation of the UI$_{E2}$ pattern for the *Borrow* event concept.

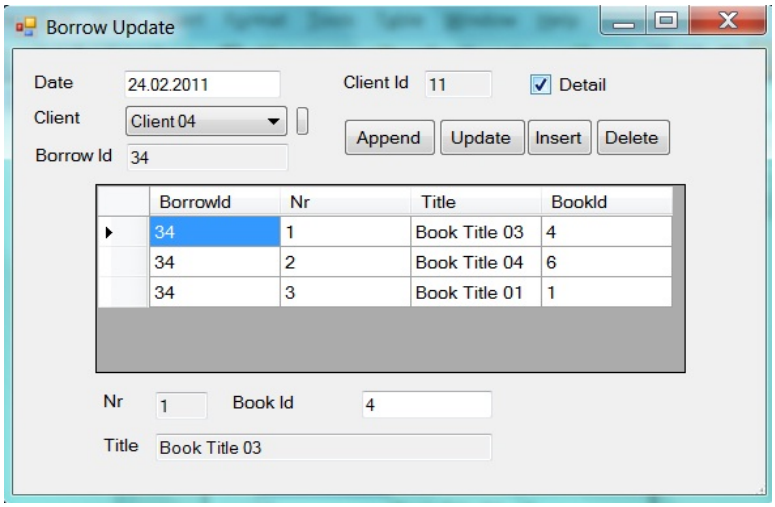

Fig. 9. An implementation of the Borrow UIE$_2$ user-interface

The UI$_{E2}$ user-interface implements the *Add* and *Modify* operations from UI$_{E1}$. In the upper left side there are the controls for editing the attributes of the

event instance. In the upper right side there are the operation controls for detailed presentation and for editing the tabular image. The tabular image contains the related instances of the BorrowDetail concept. At the bottom of the user-interface there is a single record (detailed) presentation of the current instance from the tabular image.

Fig. 10 shows the behavior of the whole UI_E user-interface pattern. UI_{E1} is responsible for viewing the instances of the event concept and the related event-detail instances. It is also responsible for deleting the event concept instances. UI_{E2} is responsible for adding a new event concept instance and the related event-detail instances, and also for editing an event concept instance and its related event-detail instances.

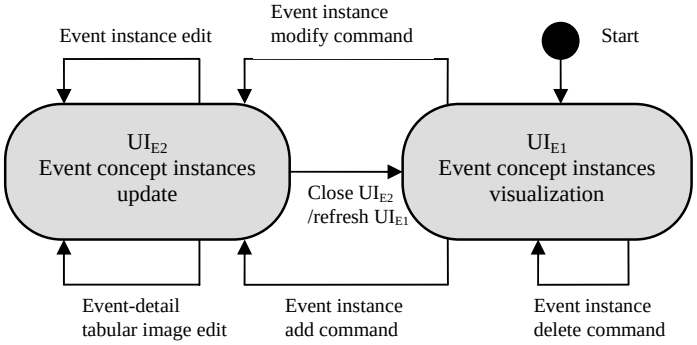

Fig. 10. The state transition diagram for the UIE user-interface pattern

5.3. A Proposal for the Resource User-Interface Pattern UI_R

The resource concept has an associated resource-archive concept. The UI_R pattern consists in two different patterns: UI_{R1} and UI_{R2} each of them responsible for some operations. The Fig. 11 presents the structure of the UI_{R1} pattern and the Fig. 13 presents the structure of the UI_{R2} pattern. The UI_{R1} pattern contains a tabular image with the instances of the resource concept, a detailed presentation of the current instance, and eight operation controls corresponding to a set of standard functions: (1) detailed visualization for the current instance, (2) synthetic presentation of the resource, (3) filter the tabular image, (4) sort the tabular image, (5) find an instance in the tabular image, (6) open the corresponding resource-archive UI_{R2} user-interface, (7) re-evaluate the resource at the current moment, (8) Select one or more field values to be transferred to a user-interface which has initiated an interaction with the current user-interface.

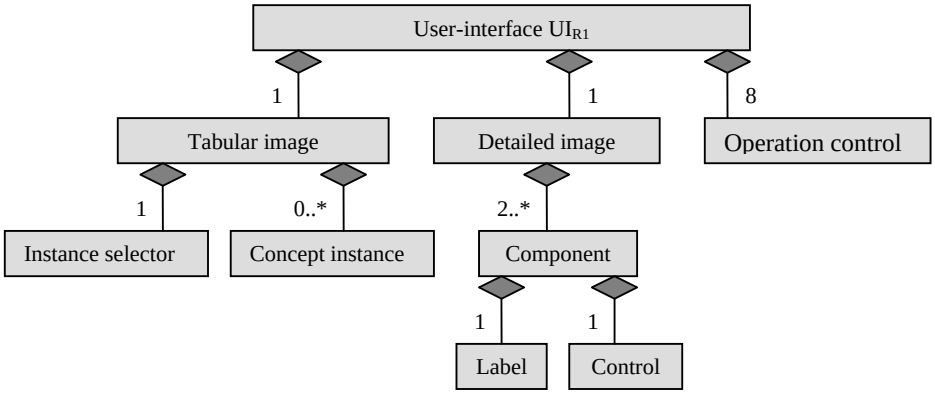

Fig. 11. The structure of the UI_{R1} pattern

Fig. 12 is an example of the implementation of the UI_{R1} pattern for the *BookStock* resource concept. The detailed visualization function is on. The upper tabular image contains the instances of the BookStock resource concept. The lower detail image shows the current instance. The *Synthetic* function shows every *BookTitle* once, summarizing the quantities. In the example of Fig. 12, if we have two books for "Book Title 10", in the synthetic presentation it appears once and a "2" value for the quantity. If we press the *Archive* button, this will open the corresponding resource-archive UI_{R2} user-interface, presented in the Fig. 13.

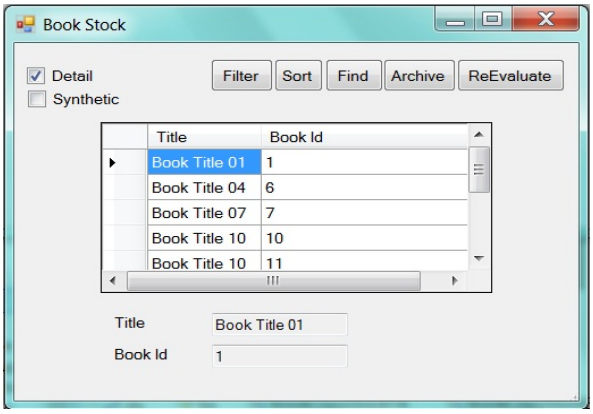

Fig. 12. An implementation of the BookStock UI_{R1} user-interface

Fig. 13 presents the structure of the second part of the UI pattern, for the resource concepts, UI_{R2}. UI_{R2} is used when we want to interact with the resource-archive instances. The UI_{R2} user-interface contains a tabular image presenting the instances of the resource-archive corresponding to the value of the *ArchiveDate*

filter control. The detailed image presents in a single record manner, the current instance from the tabular image. The nine operation controls provide a set of standard functions for the UI_{R2} user-interface: (1) the detailed presentation of the tabular image current instance, (2) filter the tabular image, (3) sort the tabular image, (4) find an instance in the tabular image, (5) re-evaluate the resource-archive at a specified moment, (6) delete the resource-archive at a given moment, (7) add a new instance of the resource-archive, (8) modify the current instance of the tabular image (9) delete the current instance of the tabular image. The operations (7), (8) and (9) are enabled only if the resource-archive corresponds to the *start moment*.

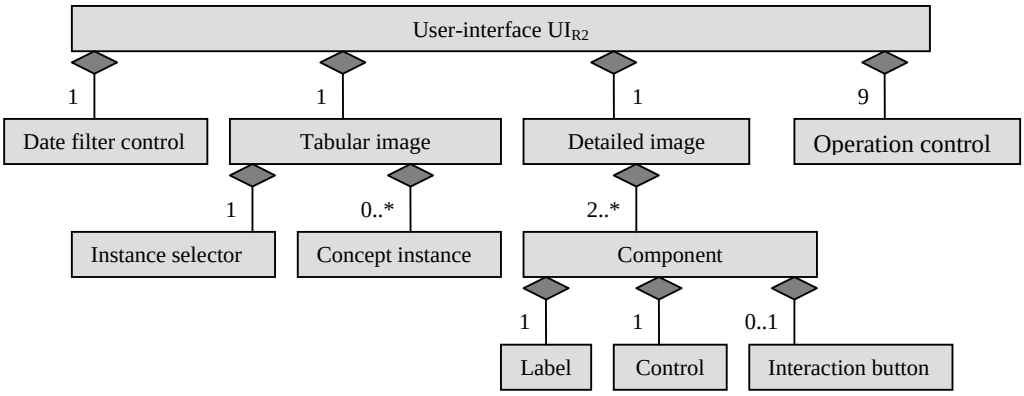

Fig. 13. The structure of the UI_{R2} pattern

Fig. 14 presents an example of an UI_{R2} implementation for the *BookStock* resource concept. The *Date* control from the upper left side enables to select an archive of the BookStock. The detailed visualization function is on. The *Filter*, *Sort* and *Find* functions are similar to those from the previous patterns. The *Evaluate archive* function enables the evaluation of a BookStock archive at a given moment. The *Delete archive* function deletes the BookStock archive corresponding to the date from the *Date* control. *Add item*, *Edit item* and *Delete item*, enable the updating of the archive from the start moment, when we started to use the library application. All other archives are not allowed to be updated, but they may be replaced by a new archive evaluation, if changes occurred in the previous archive or in the events between the previous archive date and the evaluated archive date.

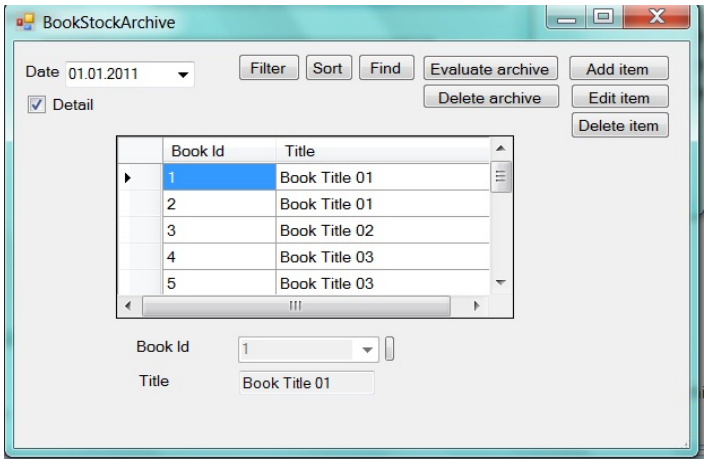

Fig. 14. An implementation of the BookStock UI_{R2} user-interface

The behavior of the UI_R pattern is shown in the Fig. 15.

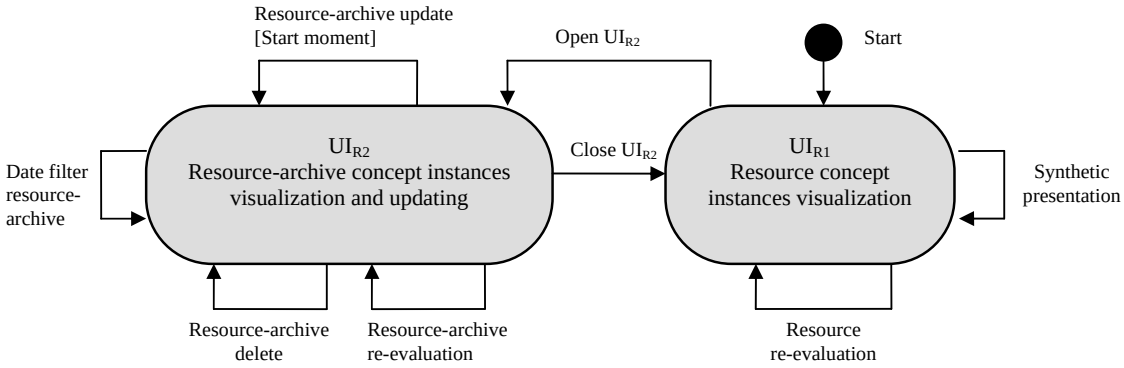

Fig. 15. The state transition diagram for the UIR user-interface pattern

5.4. The Interaction Pattern for two User-Interfaces

A user-interface of a concept may interact with a user-interface of another concept. An interaction is due to an association between the two concepts which is revealed by the conceptual model. The Fig. 16 describes a proposal for an interaction pattern between two concept user-interfaces. A client user-interface UI_{Client} initiates an interaction with a server user-interface UI_{Server} by clicking an interaction button of a control from the UI_{Client}. An object UI_{Server} is then created and activated and its *Select* operation control became visible.

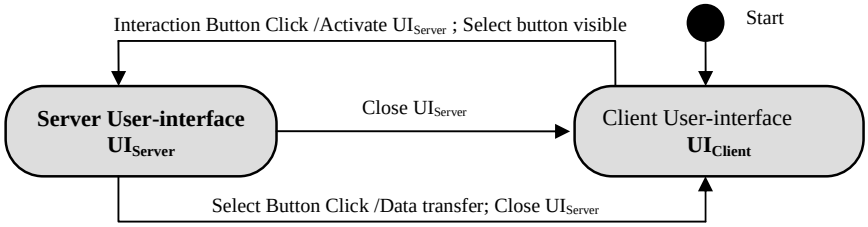

Fig. 16. The state transition diagram for the user-interfaces interaction pattern

The user works with the UI_{Server} and finaly it may close the UI_{Server} or click the *Select* operation control. In the last case, a data transfer take place, the UI_{Server} is closed and the UI_{Client} becomes active. In the same time there is a specific data transfer from the UI_{Server} to the UI_{Client}.

6. Conclusion

The proposed process follows a gradual approach, starting from a conceptual model of the application domain (1); refining the model with the REA categories and their attributes (2); generating the database schema (3); developing patterns for the UI conceptual categories and for the UI interactions (4); implementing the UI patterns for the application concepts (5). A model-based approach for the user-interface development has been used, refining a platform independent specification and progressively reaching a platform dependent specification. This is according to the software engineering techniques like Model Driven Approach [5] where the Platform Independent Models (PIMs) are refined to specific ones (PSMs) and finally are automatically or semi-automatically converted to source-code.

REFERENCES

[1] *W.E. McCarthy*, The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment, The Accounting Revew (July) 1982.

[2] *W.E.. McCarthy, L.G. Guido*. An Ontological Analysis of the Primitives of the Extended-REA Enterprise Information Architecture. The International Journal of Accounting Information Systems, 2002

[3] *P. Hruby,* Model-Driven Design Using Business Patterns, Springer-Verlag 2006

[4] *A. .Alexandrescu,* The Automate Generation of the Database Starting from a Conceptual Model, Science and Technology in the Context of Sustainable Development, Conference, Ploiesti , November 4 – 5, 2010.
[5] OMG: MDA Guide Version 1.0.1, 01.06.2003, www.omg.org/cgi-bin/doc?omg/03-06-01
[6] *M. Fowler*, UML Distilled, Addison-Wesley 2003
[7] *R. Stephens*, Visual Basic 2008 Programmer's Reference
[8] Proceedings of the Seventh International Conference on Computer-Aided Design of User Interfaces (CADUI 2008) Springer 2009
[9] CHI 2008 Workshop Proceedings, April 6th 2008, User Interface Description Languages for Next Generation User Interfaces