CookDatabase – Environment of Interactive Design and Web Applications Implementation

António Quaresma

1 IST - Instituto Superior Técnico, Lisboa, Portugal

Abstract. The information systems evolution has been a constant, and the online availability of its contents is an inescapable reality. Nowadays, it is difficult to point the limitations of web applications, since most of their diverse contents and functionalities are available and within a distance of a click. Taking this idea into account, the work developed in this dissertation tries to release quickly and consistently, for any information system based on databases, a web application that allows manipulating its contents. At the same time, it provides the possibility of an authenticated user to develop its own information system, through a web interface. As result of the researching analysis and developing requirements of the work, it was selected the Execution Model in order to make the desired development, thus giving continuity to the work outlined in the CodeExecutor [1].

Keywords: Model-Driven Architecture, Domain-Specific Modeling, Execution Model, Patterns, Parameterization, Parsing, Metamodel and Data Base.

1 Introduction

The Web and systems applications have been highly used, for both personal and commercial aims. Such generalized use, allows that an increasing number of people are adapted to the interface of a browser, creating a great potential for this type of applications, since they are adapted for a wide range of users, in principle [6], [3], [4].

The development of a web application is characterized for a high number of identical functions, which can happens within the same applications and between different applications. It is easy to note that the way of browsing, presenting the data to a user, and manipulating the information existent in the system is, in most cases, identical according to the application. While this not happen to all the applications, this is true for many of the cases [6], [5].

Taking this perspective into account, it is not surprising to verify that most of the actual web applications have generated code [6], [7].

Currently, there is no consensus about the way of generating code, and the best way to do so. Two of the most relevant approaches in this area are its generation trough a model (Model-Driven) or through a domain (Domain-Specific). As expected, the generated code is repetitive. If we are generating the code for browsing in a system based on a Database (DB), it is expected that the various pages allowing the access to each one of the existing tables have the same code, where the unique difference is on
the selected table and browsing. Nevertheless, since this code was generated without effort, there is not, in fact, the problem of being repetitive, but at the level of its maintenance and modification. If there is a lot of code, it is necessary to replicate one alteration for the entire code. In this case, it is important that the generation system runs such changes, even if it is necessary to re-generate the code again. Such situation creates other problem: not all changes or customizations can be parameterized, because when developing the code generation for all requirements of an application, it would complicate so much the parameterization and the necessary configurations that would be easier to create the code from the root. Taking into account the characteristics described above, it is possible to verify that, in the same way that we generate code, and because this is always similar, it is also viable to take a different approach, the code execution.

In practice, the objective is in everything identical to the generation code, but rather than generating this one repeatedly to perform the same operations, and for each one of the pages of a web application, the objective is to develop it by following the perspective of a system. In other words, there is a given code that, dependent on the receiving inputs, returns different outputs. For instance, to list the tables of a BD, instead of generating code for each one, there is a unique code that receives the name of the table, where the result is a web site with a list of data that exist on the referred table.

1.1 Objectives

This work has as main objective the development of a web application which is based on web patterns, i.e. listing, editing; allowing the interpretation of a DB of any information system, filling the necessary metamodel for its proper operation and providing forms to the user in order to manipulate the records/objects of the system. The mentioned work arises from the adoption of the application CodeExecuter [1], which makes a fraction of the desired parameterization according to the text files. Initially, the objective consists in passing the parameterization to a relational support, through the construction of a suitable model to be introduced in the DB. Once this step is completed, the constructed application will be able to read the parameters from the DB, which are related with a particular table/class, constructing according to them the appropriate forms. Following the parameterization consolidation from the DB, the work enters the second phase, which extends the capabilities of the parameterization. The purpose of this phase passes by dealing with complex relations, such as hierarchical tables, but also by the parameterization of the graphical appearance of the generated forms.
In a third phase, it is pretended to present the users with a way to develop new DB, which after created can be manipulated through the developed work in both phases one and two. The architecture of the work done, related with the CookDatabase application, is represented schematically in Figure 1.
2 CookDatabase – Developed Patterns

2.1 CookDatabase – Patterns

The CookDatabase is developed over the execution model [14]. In order to apply the referred model it was done a survey about the patterns, which is necessary for browsing and manipulating the records/objects from a DB of any IS, and then it was developed the code to support its execution.

2.2 Generic Patterns

As suggested by the term itself, the generic patterns aim to be generalists, and within the context of this work this means to support the browsing and editing of a DB of any IS. This is one advantage, but one limitation also, since this kind of patterns can not represent the whole particularities of an IS. Next, the generic patterns identified/developed are presented:
- **ListBegin** – Consists in the home page of the application, appears after the authentication made by the user, and after this one choose the intended application to browse. Here are listed the various possibilities that allow the choice of the input point in the actual IS (Figure 2). Selecting one of the available options, the user is directed for the next pattern, the ListingEdit.

![ListBegin Pattern](image1.png)

Figure 2. ListBegin Pattern.

- **ListingEdit** – Pattern used to list the entities of an IS. For each one of the entities accessed by the user, it is created one table with the attributes of the entity of the IS, with its respective headers and contents. Moreover, links are created in each one of the table records, allowing the editing and deleting of themselves. Finally, other links for the entities related with the table are created also (Figure 3).

![ListingEdit Pattern](image2.png)

Figure 3. ListingEdit Pattern.

- **FormEdit** – Edition pattern of the IS entities. After a user click the edition link in the ListingEdit pattern, is directed for the FormEdit, where there is a form with the selected entity fields available for edition. This pattern is used to create and
delete entities: when deleting, the entity attributes are filled with the record values existent to delete; and when creating, the attributes are empty (Figure 4).

Figure 4. FormEdit Pattern.

- FormRoot – Pattern that extend from the FormEdit, it is particularly used when the entity being selected does not have any reference to other table. Its use is necessary since it simplifies the development, when one entity does not have a foreign key to other entity.
- ListingSearch – Related with the pattern that provides the user one way of searching about any entity attribute (Figure 5).

Figure 5. ListingSearch Pattern.

2.3 Specific Patterns

The specific patterns represent patterns occurring within a given application, not in all web applications based on DB, as it happens with the generic patterns. These patterns inherit from the generic patterns and add features to optimize the manipulation of data for a particular IS. They serve as an extension to the application but are not necessary for its proper operation. For the CKDB, more specifically to one of its components (DBManager), two patterns were developed to improve the interaction between the user and the creation of attributes and restrictions of the DB being developed.
- **Attribute_FormEdit** – Pattern that extends from the FormEdit, which is only used by one of the application components, the DBManager. This pattern optimizes the creation of new attributes, managing the fields that are needed to fill up, depending on the type of attribute that the user will create (Figure 6).

![Figure 6. Attribute_FormEdit pattern.](image)

- **Constraints_FormEdit** – Pattern that is one specialization of the FormEdit also, which is used to create restrictions or relations between entities (Figure 7).

![Figure 7. Constraints_FormEdit Pattern.](image)
3 Components & Workflow

The CookDatabase is composed by two components, which are described below:

- **DBBrowser**, allows the user to interact over any DB interpreted and loaded in the MT-CKDB.
- **DBManager**, offers to the user the possibility of creating a DB using as interface the DBBrowser, i.e. functions as an extension of it.

### 3.1 DBBrowser

The DBBrowser component allows the user to browse/interact over any DB loaded in the CKDB metamodel [14]. The user interacts with the DB of an IS using the generic patterns (Chapter 2, Section 2), listing and manipulating its entities, thus having the possibility to change the contents. This component aims to represent the DB hierarchy, as well as the manner that the tables are linked.

### 3.2 DBManager

The DBManager extends the DBBrowser, since it uses the same patterns and, additionally, the specific patterns (Chapter 2, Section 3). The objective of this component is to offer a web application for the development and maintenance of DB, which are loaded in the metamodel of the CKDB [14]. So, the user can create new tables, attributes, relations and restrictions. At any time the user can visualize the list of all desired changes, and choose accepting or canceling such alterations. This component represents the system tables of a DB, where the tables’ information is kept, as well as the attributes, relations and views.

### 3.3 Workflow

As referred above, the CKDB allows the user to effectuate various actions over the system. The group of these actions is characterized by a workflow, which is described in the present section.

The CKDB workflow is composed by four steps, divided into two components. The DBBrowser component uses the two first steps, Reverse and Interact, and the DBManager component the following two steps, Forward and Round-trip.

- **Reverse**, performs the parsing of a DB of any IS, and loads that information in the MT-CKDB (Figure 8). The MT-CKDB comprises a group of procedures (DBParsing) able to interpret a DB and loading the necessary information for the tables’ parameterization.
• Interact, allows the user to manipulate the objects/records of the DB that was interpreted in the previously step (Figure 9). The patterns manipulate the DB of the IS that was loaded on the Reverse process through the information that is contained in the MT-CKDB.

• Forward, allows the user to develop one DB, building entities, attributes and relations, and at any time the user can generate the developed DB (Figure 10).
Round-trip, after the user has developed and generated its DB in the DBManager, now has the possibility of actualize the model developed by himself, seeing the actualizations replicated, either on the DB or on the DBManager (Figure 11).
Figure 11. Demonstration of the round-trip process of the workflow.
4 Conclusions

4.1 Conclusion

In Chapter 1 provides a brief explanation about the research work done in this paper. It describes the execution model, its goal and the guidelines for the developed work.

Chapter 2 intends to present the two types of developed patterns, which are briefly described next:

- generic patterns, which are used in all applications based on DB and,
- specific patterns, which are used for a given application, representing behaviors which are repeated throughout this one.

Additionally, Chapter 3 provides the presentation of the two developed components:

- DBBrowser, which represents the hierarchy of a DB, supports the parameterization and allows the system to load the necessary information to apply the patterns and,
- DBManager, which represents the structure of a DB, allowing any user the development and maintenance of a DB.

Finally, the referred chapter provides a description of the various workflow processes, which allows interpreting a DB of any IS, and load it in the metamodel, browsing over the loaded DB and manipulating their data, developing a new DB and actualizing the developed DB.

4.2 Future Work

Following the development step of the CookDatabase and description of the possibilities that it offers, becomes relevant to outline some considerations in terms of future work. In this sense, there are two questions that should be discussed:

- the improvement that can be applied into the CookDatabase and,
- the innovation added, how far we can take with the kind of approaches presented in this work.

In relation to the first question there are several points deserving a reflection:

1. Management of the permissions for the created users, at the time of the generation of a new DB in the DBManager component. Now, such permissions are very general, thus allowing the CookDatabase administrator to effectuate their management would be a real asset.

2. Download of the generated scripts, throughout the application there are always scripts being generated, either for the creation of a DB or for its actualization. Although the possibility of downloading the script of the DB creation, both update and users creation scripts are not available. So, would be interesting its availability for downloading and the possibility of the user to choose between running them from the application or execute the download to run when intended.
3. The CodeExecuter development [1] is done over the .Net framework 1.1 [8], whole the reused code throughout the development was updated to run in the framework 3.5 [8]. So, there are some functionalities that should be explored, such as:
   3.1. the ObjectDataSource use [8]
   3.2. the AJAX controls, (e.g. UpdatePanel)
   3.3. both use of the DynamicGridView and DynamicFormView related with the ObjectDataSource.

In the future, and considering the innovative question, there are various aspects that deserve attention in terms of development:
   1. Integration with the GenericWebForms [9] for the creation of forms.
   2. The webparts use [10], for the creation of forms, after the referred integration performed.
   3. The webparts use for patterns development and costumization.
   4. The DB development through a web application, like DBManager, but modeling. The tables and their attributes would be webparts, which can be dragged into the application working area and linked for developing the DB.

The complexity level, and the web applications possibilities, is growing and its potential for growth is endless since, and increasingly, any type of functionality is available on a web application. For instance, a wide variety of applications such as iGoogle [11], website to costumize the creation of pages with the most diverse features; Microsoft Office Sharepoint [12], collaborative websites allowing the portals development, Adobe PhotoShop [13], website with tools for editing images.
References


http://www.google.com/ig

http://www.microsoft.com/Sharepoint/default.mspx

https://www.photoshop.com/express/landing.html