Dynamic Code Generation for Cultural Content Management

Maria Giatsoglou*, Vasiliki Koutsonikola*, Konstantinos Stamos*, Athina Vakali* and Christos Zigkolis*
*Aristotle University of Thessaloniki
Email: (mgiatsog, vkoutson, kstamos, avakali, chzigkol)@csd.auth.gr

***Abstract*—Digital repositories are popular means for preserving, restoring, and indexing archaeological and cultural content. They provide the base for development of a fauna of related applications including virtual tours and data management. Common difficulties such as the ever changing software specifications from domain experts make this task challenging as the alterations of the database schema lead to massive code rewrites. Within this context we propose and implement in practice a model for automated code generation building essentially a content management application by traversing a custom tree-based ER-schema.**

I. Introduction

Digital representation of cultural content is a popular trend followed by an abundant number of museums worldwide. In that way not only the cultural content is preserved, but it can be indexed, categorized, browsed and exposed to a wider public which cannot be physically present at the museum. The digital representation is usually the basis for the development of high level services providing virtual tours to the visitors.

The basic steps of digital representation include the digitization of an item (e.g. a photograph of a jar), the gathering of related information (e.g. dimensions of the jar, historical facts, etc.), and finally storing into a database. Apart from the archeologists' tasks of digitization and information gathering, the design of the database repository itself and the set of applications for data entry and retrieval is a tedious and challenging task. The primary issues that a computer scientist has to face are:

- Data interpretation between the domains of archeology and computer science. Archaeologists usually use language that is domain specific to express the attributes of the items while the computer scientists fail to understand properly what is required. Usually a set of interviews deal with this issue as part of a software development process [1].
- The ever changing database schema. New attributes are usually added to the items, others are deleted or changed, and tables are created and deleted throughout the development cycle.

Both issues lead to additional development cycles, coding effort, project time scheduling problems and in general the developed process is ill mannered. The source code tends to include specific exceptions and hard-coded parts as quick fixes to the ever changing specifications. Additionally, the source code becomes repetitive in many modules as there are common patterns for handling database tables and generating forms for CRUD (Create Retrieve Update Delete) services.

In this context we face the problems related to the design of a database repository for cultural content and the development of related software for data entry and retrieval. Our primary contributions are:

- Suggest a generic tree-like E-R diagram suitable for cultural content.
- Propose an algorithm for dynamic code generation for CRUD web applications.
- Tailor the MVC (Model View Controller) [2] in the code generation process.
- Implement a complete model that incorporates the concept of building cultural content management applications dynamically.

Our model has been used in practice on behalf of the Athanasakeion Museum of Volos (Greece) with success. We believe that this paradigm can be of great value to similar in nature situations of software development.

In Section 2 we present the model for dynamic code generation while in Section 3 we discuss its advantages and disadvantages. Finally in Section 4 we conclude.

II. Proposed Model

Here we provide an in-depth description of the proposed model and its various components. as seen in Figure 1 it consists of three well separated tiers:

1) ER-schema. In this tier resides the Entity-Relationship schema [3] implemented in the form of a traditional relational database. All the cultural content is stored and organized here.
2) Code Generation. It includes a component that parses the E-R schema and taking into account several constraints generates source code.
3) Application Services. The generated source code is deployed on-line and consists a web application that offers CRUD services for the database.

Each individual tier is comprised by several software modules that interact with each other providing a streamlined operation for code generation and deployment.

A. ER-schema

In terms of the database schema, the cultural content exhibits two types of organization; the *physical organization* and

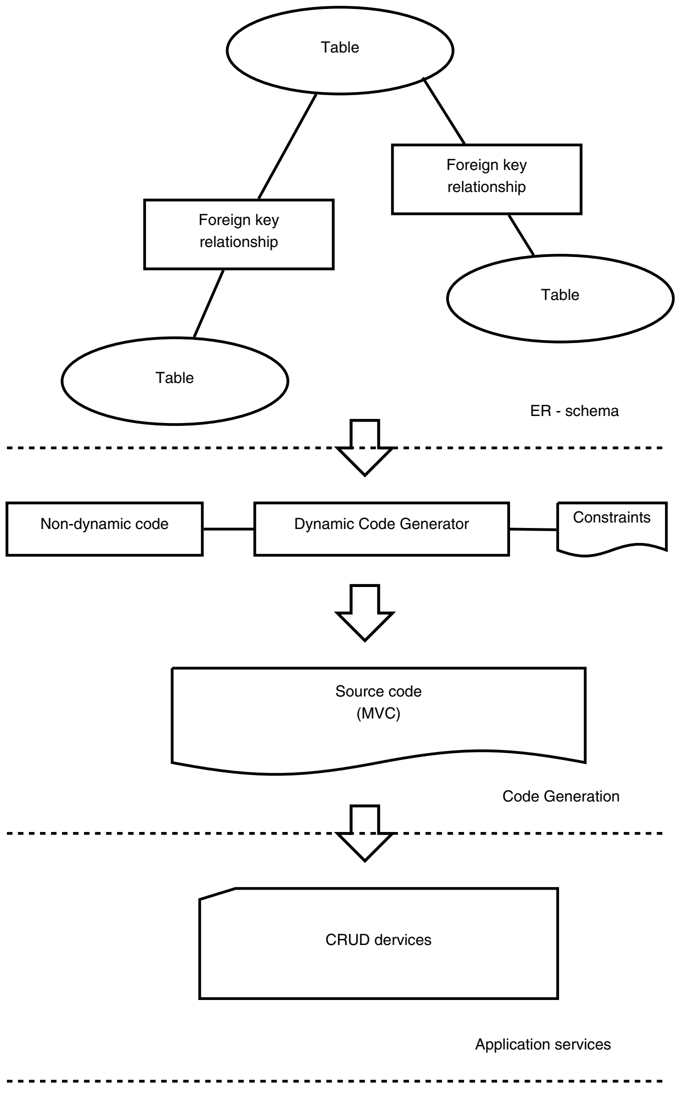


Fig. 1. Proposed Model

the *logical organization*. The physical organization refers to the spatial arrangement inside of a museum or an archaeological site of the items. An item is usually part of a collection organized in a showcase where the showcases are placed inside a room. Finally the rooms themselves are part of a complex building structure as a museum. The logical organization is done through a hierarchy of item groups under specific thematic areas (i.e. stone age tools, stones age weapons, etc.). As a result, in both organization types we see that in a bottom-up manner we begin by leaf level simple entities up to higher level grouping.

Each item is described by several attributes such as dimensions, materials, descriptions, pictures, etc. Each individual attribute may have a more complex representation, for instance, a description contains text of specific language and audience. Therefore, we identify the *shallow view* of an item and the *deep view* for each complex attribute.

From the above discussion we conclude that a tree representation of the database base schema is suitable for describing the items. We are focused on a 6NF [4] with every table being related to other via foreign keys as seen in Figure 1. The advantage of this tree representation is that we can parse the tree and produce nested views for each table as part of the code generation process.

B. Code Generation

The code generation is the core of the system that implements the logic of the generated application. It parses the database schema and generates code for each table using the MVC design pattern. The generated code is deployed at a Web server creating effectively a Web application for CRUD operations [5].

The MVC design pattern is suitable for applications that contain a mixture of data access code, business logic code, and presentation code. Such applications are very difficult to maintain, because changes to the initial data representation (as in our case) cause a strong ripple effect to the code. A common pitfall is the strongly coupled code for access, logic and presentation which leads to redundant coding, copy-paste tendencies and difficult maintenance. Consequently, we designed our code generator to perform such a decoupling procedural according to the MVC pattern.

The key challenge of the code generator is to identify common or frequently occurring patterns in the required code. This careful inspection leads to a lightweight and elegant design that demands minimal coding effort of elemental particles of code to be reproduced in the final output. The generator uses the Algorithm 1 for creating the Views for each table in the database. The Models and the Controllers are designed in a similar manner. The algorithm produces recursively a Form F that starts from a given table T, for a specific CRUD service S, given several constraints C, and static code H. For example, the T can be a table for the material types, S can be the Create operation, C the constraints that a material cannot have a number as name, and H a specific header for CSS. Then, the algorithm starts from T and recursively generates code for the *shallow view* and creates nested placeholders for the *deep views* of the related tables. The algorithm essentially parses the tree of relations starting from the parent table T until leaf level.

C. Application Services

The generated code consists a complete Web application that provides CRUD operations. For each table in the database one may create a new entry, retrieve it, update, and delete it. The main difference from traditional scaffolding tools is that if the user select a complex entity all the individual parts that assembles it are also available in the same Form in a nested manner with CRUD operations available.

An important characteristic is the flexibly of the Retrieve operation thanks to the well-shaped database schema. The user may enter a few keywords, and a Depth-First Search is performed in the entire database tree. Therefore it is not necessary to know exactly what the user is looking for, any

Algorithm 1 Synthetic Form Generation

```
Input: $T$, the parent database table
Input: $S$, service type (Create, Retrieve, Update, Delete)
Input: $C$, constraints
Input: $H$, static code for headers
Output: $F$ a synthetic form
1: $F$ = new form including $H$
2: $t = T$
3: $p$ = new placeholder in $F$
4: repeat
5:     create shallow form for every column of $t$ in $p$ according
       to $S$ and $C$
6:     $p$ = new placeholder in $p$
7:     $t = t -> next$
8: until $t$
```

field in the corresponding table could match as sub-string or any other of the children nodes recursively.

III. Discussion

In the previous section we suggested a model to deal with the changing specifications in the paradigm of cultural content management. In this section we identify the strengths and weaknesses of that model and provide implementation details in practice.

A. Advantages

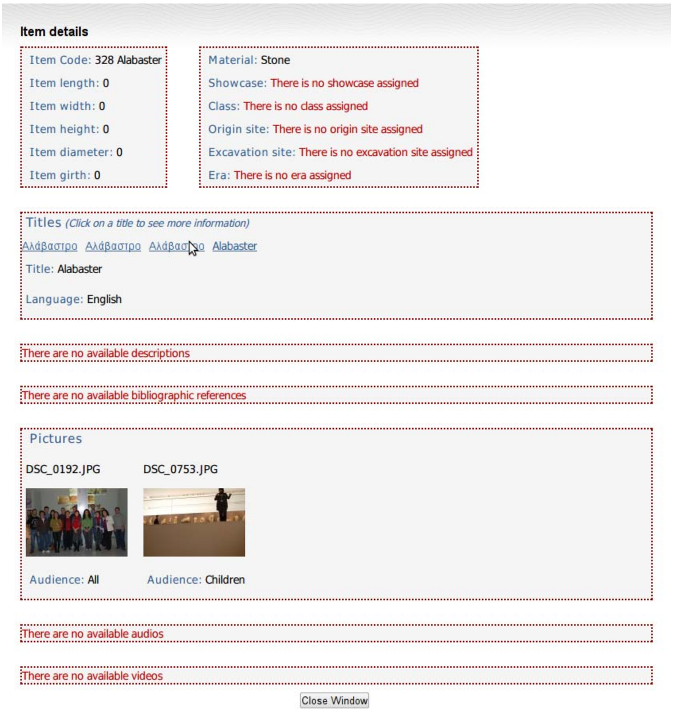
The benefits of the model emerge in the long-term, as the initial specifications change constantly and the initial assumptions about the data are proven false. In particular we identify the following strong points of the model:

- Single point of failure. If at any point an undesired behavior is detected or a bug is found, there is a single place in which the developers will focus. This is the code generator and especially where the code templates are defined.
- Minimize code replication. Coupled with the previous point, the developers may fix a bug in a single location and not in different replicated places. Furthermore, replication minimization greatly reduces the development time leading to a more elegant coding base.
- No hard-coded code. Everything is described as templates and elemental particles of code enabling the ease of change in the code.
- Auto-adaptation. Changes to the underlying schema are almost transparent to the rest of the model. The database schema is parsed dynamically with no assumptions about the nature of the data itself as long as the tree-based structure is respected.

B. Disadvantages

The tradeoffs of having a dynamic coding system are related mostly to the difficulty of the endeavor. More specifically:

- Deep domain understanding. Perhaps the most difficult part if to comprehend in depth domain specific data and extract the necessary and usable information. A careful analysis is required in order to transform them into a proper tree-based representation.
- Common coding patterns detection. Another procedure that requires skill is the full analysis of the problem and the design of a detailed description of the final application. This is an essential part of finding common and frequently required code patterns and operations.
- Coding difficulty. A final challenge is the coding effort that is required. In the traditional case one would copy-pate common code-parts leading to a bloated code-base. Sacrificing and initial speed in development one is required to perform a template based generic coding which will payoff later



Item details

Item Code: 328 Alabaster
Item length: 0
Item width: 0
Item height: 0
Item diameter: 0
Item girth: 0

Material: Stone
Showcase: There is no showcase assigned
Class: There is no class assigned
Origin site: There is no origin site assigned
Excavation site: There is no excavation site assigned
Era: There is no era assigned

Titles *(Click on a title to see more information)*
Αλάβαστρο Αλάβαστρο Αλάβαστρο Alabaster
Title: Alabaster
Language: English

There are no available descriptions

There are no available bibliographic references

Pictures
DSC_0192.JPG DSC_0753.JPG
Audience: All Audience: Children

There are no available audios

There are no available videos

Fig. 2. Synthetic Form

C. In practice

The model has been implemented in actual code and it is currently running at the Athanasakeion Museum of Volos. The database repository contains cultural data from Neolithic artifacts and it is implemented in MySql. The code generator parses the database schema by querying the MySql database and generates a Web Application for the CRUD operations and it is written in Python. The generated Web Application is implemented in PhP for the server-side and in Javascript and CSS for the client-side. The Non-dynamic parts of code (H) include headers related to CSS, scripts, and cookies for log-in session management. A preview of a Form that displays data for an item is illustrated in Figure 2. The various placeholders are clearly visible as produced by Algorithm 1.

IV. Conclusion

In this paper we surveyed a set of issues related to the design and implementation of a database repository for cultural content and its accompanying Web application for data entry. Following a bottom-up approach we suggested a tree-based structure for the ER-diagram which enables code generation via an Algorithm that uses nested views of a table. The final outcome is a Web Application for data-entry. The suggested model is suitable and adapts automatically to changes in the ER-diagram. It has been implemented in practice and it is part of a day-to-day data entry from archaeologists. We believe that our approach is of potential use to software developers working on related paradigms.

Acknowledgment

This work was supported by Iceland, Lichtenstein and Norway through the EEA Financial mechanism under Grant EL0043

References

[1] L. Rising and N. S. Janoff, “The scrum software development process for small teams,” *IEEE Software*, vol. 17, pp. 26–32, 2000.

[2] A. Leff and J. T. Rayfield, “Web-application development using the model/view/controller design pattern,” *Enterprise Distributed Object Computing Conference, IEEE International*, vol. 0, p. 0118, 2001.

[3] P. P.-S. Chen, “The entity-relationship model—toward a unified view of data,” *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, 1976.

[4] C. Date and H. Darwen, *Temporal Data and the Relational Model*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[5] H. Kilov, *Business Specifications: The Key to Successful Software Engineering*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.