

ArtSite - a new approach to create and manage modern websites using the Relational Data Base Management Systems *

Natalie Zubkova[†]
Arttechnics.com
4.8-1, Teterensky Lane,
Moscow, 109004, Russia

NatalieZubkova @arttechnics.com

Eugene Titov[‡]
Pennsylvania State University
135 Hammond Building
University Park, PA 16802, USA
Evt101@psu.edu

ABSTRACT

In this paper we present a new approach to the development and management processes of a modern website. The crucial idea of the proposed method is to use the Relational Data Base Management Systems (RDBMS) not only to store the data of the websites, but also as a tool to organize the website development processes.

Categories and Subject Descriptors

H3.5 [Informational Systems]: Informational Storage and Retrieval - Online Information Services

General Terms

Content Management System

Keywords

Content management, Web services, Website development

1. INTRODUCTION

Creating and managing the contents of the modern websites, along with the development of the core structure of the websites, poses significant challenges, which have been addressed in a number of Content Management Systems (CMS) presented by the software developers such as IBM [1]. Interesting research papers have been also presented recently [2, 3]. The current approach is to integrate databases into the web environment [4], and the current trend is to use the relational data base management systems or object-relational database management systems (ORDBMS) as a platform to maintain the websites' contents.

*Presents formal definition of the proposed model and examples of implementation.

[†]Arttechnics.com, CEO.

[‡]Arttechnics.com, Co-founder.

The groups of specialist involved into a modern website development process are presented, for example in [1], these groups are:

- system architects (who creates initial software design and database structure);
- programmers (who implements the software design);
- designers (who develops the website style and implements it as HTML or XML pages);
- usability engineers (who helps identify problems with the user interface);
- information architects (who designs document types);
- business developers (who develops the business case)
- project managers (who oversee the deployment);
- content managers (who create new content and migrate existing);
- system administrators (who create user profiles, make backups etc.).

Simultaneous work of these specialists, faces known website management problems, some of them are:

- lack of separation between a website general logic (design) and its software code and as a consequence a strong dependence of the website architects and business case developers from the programmers, as well as a dependence of the content managers from the designers, along the whole project life cycle;
- low ability of a simultaneous work of different specialists such as designers, database developers, and application programmers during the website development and support stages.

In this respect, we propose a conceptual model of a website creation procedure, which takes advantage of the RDBMS generic capabilities in order to overcome the website creation and maintenance problems mentioned above. In particular

we propose to use the RDBMS to maintain the access control rules, which would provide a simultaneous but safe access to the development instruments, website structure components, and the website contents, by the groups of specialists, involved into the process.

The paper is organized as follows: in section 2 we discuss the general logic of the *Artsite* model and the data types managed by the proposed tools implementing the model. In section 3 we introduce the software modules necessary to implement the model and provide some examples of the web pages created using the instruments of the *ArtSiteConstructor*, a software tool which implements the logic of the presented model. Final conclusions are presented in the chapter 4 along with the ideas of the future work.

2. ARTSITE MODEL

2.1 General idea

In our model we introduce the *blocks* as the entities upon which a website is constructed. The HTML part of such a *block* implementation, is unique no matter how often it is used to construct parts of different web pages. In this respect the idea is close to one implemented in the *shtml* model. However the code (for example Java or PHP) associated with each appearance of such a *block* in a web page, can be different and so can be the queries (for example SQL), which retrieve the data, for each particular instance of the *block*. In the other worlds, we consider the data (contents), meta data (HTML), logic (Java or any other language) and the queries (SQL), forming the instances of the *blocks* as independently managed objects. All these objects are still building bricks of the *blocks*, but they can be organized by means of the Relational Data Model and therefore managed by a RDBMS. While the final representation (a generated page) can be formed as an XML code, the underlying data model is quite different, because initially the logic (Java code in one of the realizations described in the section 3) is stored and managed not in the special tags but in the database records.

The following types of the *blocks* are used in the model:

- a regular *block* - an entity which is a container for other *blocks*. It consists of the header and footer (both are sets of HTML tags);
- a *template* - the *block* which consists of the header, the footer and the body. All these three fields are pieces of the HTML code which includes special tags, defined in the *Artsite* model;
- a function of type *condition* - the *block* with associated code which returns "true" or "false" depending on the initial data, results of the database queries and the function logic.
- a function of type *cycle* - the *block* with associated code which prepares and returns a data array.

2.2 The data structure

Once the definition of the *block* is provided, the following step is to describe the methods which are used to construct a website pages using the *blocks*.

We first present the relational data model necessary to support the proposed approach, and then describe the algorithm. The simplified data model is presented in the Fig. 1.

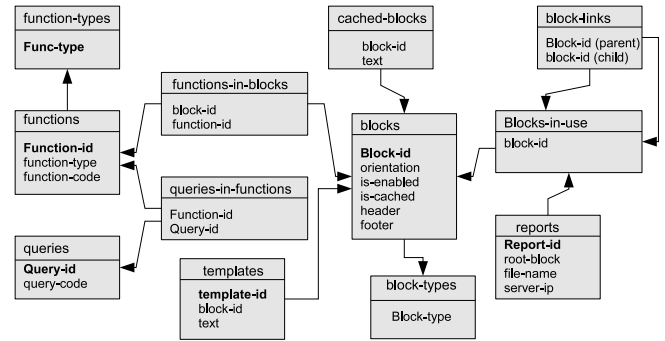


Figure 1: The Artsite data model.

Following relations and relationships are presented in the figure:

- *blocks* - the relation which stores the *block* attributes. The important attributes are: 1) orientation - the way in which children *blocks* will be presented in the web page (we use the hierarchy of *blocks* as the main way to construct the pages); 2) is-enabled - some of the *blocks* may not appear in the pages; 3) is-cached - the *blocks* with the static contents (but not the whole pages!) can be cached to improve the system performance; 4) header and footer - the attributes in which the starting and ending HTML tags of the *blocks* are stored.
- *block-types* - specifies what kind of *block* (*regular block*, *template* or *function* each of the *blocks* is);
- *templates* - a relation to store the bulk HTML code of the *blocks* with special to *Artsite* tags in it;
- *functions* - a relation to store the code of the *functions*. Two types, of currently supported *functions* are: *cycles* and *conditions*. The purpose of these *functions* will became clear latter, when the algorithm of the *parser* which, creates the final HTML code of the pages, will be explained. The function types are stored in the relation *function-types*;
- *blocks-in-use* - the relation to store instances of the *blocks* (their appearances in the webpages);
- *block-links* - the relation to support the *block* hierarchy.
- *cached-blocks* - the relation to store the final representations of those *blocks*, which contents is static (or updated by a regular procedure for example based on a schedule).

2.3 The algorithm

The general algorithm is presented in the Fig. 2. Several procedures, such as the analysis of the *block* state (cached or not) are excluded from the picture for the simplicity. Before the algorithm starts all of the *function* codes, stored in the database, are to be compiled, and stored on the disk, as a set of executables or class-files, if languages such as Java are used to write the *functions*. If the *functions* are written in script languages such as PHP, they can be started directly from the database. When a page request is transferred from a web server to the system main engine (we can use servlets or PHP bridges to do so), the page *parser* is involved to create the final presentations of the requested web page (its HTML code).

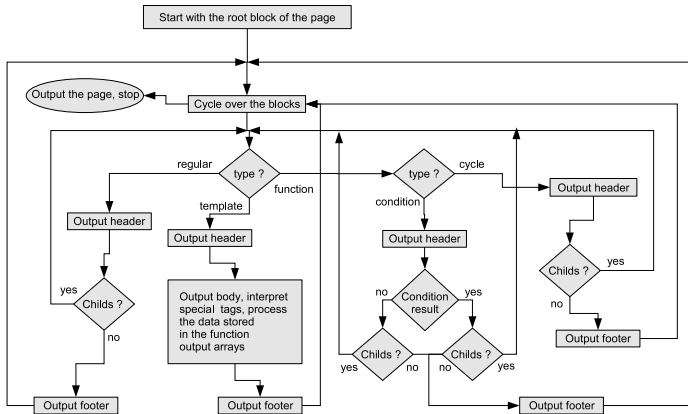


Figure 2: The Artsite algorithm.

The parsing process starts with the root *block*. The identifier of the root *block* is received by the *parser* as an argument. Starting from the root, we process each of the *blocks* according the hierarchy upon which the page is constructed.

We first analyze if the current *block* is cached, and if so we simply output its contents and proceed with the next *block*.

For each of the *blocks* we first output its header. If the *block* is a *regular block* or a *template* and it is a child of a *block*, which is of type *cycle* (according to the *function* type associated with the *block*) then its header (and body in the case of *templates*) can contain the special (to *Artsite*) tags, such as "column=N". These tags specify that the function of the parent *block* produces some data (for example the result of the database query) which need to be outputted in the final representation of the child *block*.

If the current *block* is of type *function*, then, based on the request arguments, the corresponding *function* analyze the incoming data, prepare the values for variables of the database queries and evaluate the queries.

If the *block* function type is *cycle*, then the query results are stored into the array, which always have the same name and structure according to the system agreements. The data from this array is used to process the "column=N" special tags, where the N simply means an index in the output array, produced by the *function* of the type *cycle*.

If the *block* function type is *condition* then instead of the data array, the *function* associated with the *block* produces an output code which has two distinct values (true or false). Depending on these values one or another child *block* is used to prepare the final HTML presentation of the *block*.

After all the necessary data is stored into the array, or the return code of the *block* is produced, the *parser* continues building the final representation of the *block*, taking into account the HTML code associated with the footer of the *block*, the data in the output array, or the *block* function return code.

The process continues until all the hierarchy is processed and the whole HTML presentation of the web page is build.

Following the algorithm, a web page is dynamically created. In this process the main data sources to create the page are:

- the logic of the page (presented as the combinations of *blocks*);
- the data from the databases;
- the data from the cached *blocks*;
- metadata and other data from the HTML code associated with the *blocks*.

2.4 Website development

Since the *blocks* and their attributes are stored in the database, they can be easily used to build all of the pages necessary for the web project.

Once the necessary tools (such as *PageConstructor* described further in the article) are available to construct the pages, the construction process is as follows:

- create the general logic of the web pages (as a scheme), analyze the number and types of the unique *blocks* necessary to create the web pages;
- create the visual representation of pages (their HTML code) as combinations of the pure HTML parts of the *blocks*. Each of the *blocks* is thought to be unique, if its outlook is unique, no matter how often it is used to construct different pages;
- Prepare the project related database;
- Write the *block function* codes and database queries, take into account the fact that the same *functions* and queries can be related to different *blocks*.

2.5 The roles

As was mentioned in the introduction to the paper, one of the challenges, during a website development stage is to support the simultaneous work of the specialists involved into the development process.

Since we handle the HTML code of the *blocks* as well as their *function* codes and queries as database objects, and also the very page structure is a database object too, we can

easily develop the access control rules which will support the different roles.

In other words, each of the groups of developers, involved into the process, will get access only to the data components, they really need to access.

- The *architects* and *business rules developers* can construct the general logic of the pages (using the *blocks* and relationships between them) not taking into account details of both HTML and (for example) Java implementations of the *blocks*.
- The *designers* will only work with the HTML code while the system will protect the software code associated with the *blocks* from an accidental damage.
- The software code is accessed by the *programmers* only.
- The *data base developers* are working on the data-base scheme, its implementation, and the queries, using their own tools.
- The actual data, associated with the *blocks*, is stored in the project related databases, and therefore can be accessed by the *content managers* independently, using project related interfaces, or general content management systems in the simple cases.

3. IMPLEMENTATION EXAMPLE

Another way to describe the proposed approach is to discuss the functionality of the *ArtSiteConstructor*, - a software package, which is based on the ideas of *Artsite* - a general approach under the consideration in this article. The implementation uses PHP bridges to transfer the data from the Apache webserver to the system engine, written in Java. The Java language is also supported to write the *block function* codes of types *cycle* and *condition* as presented in the previous chapter.

The *ArtSiteConstructor* is a typical client/server software.

The client consists of the *PageConstructor* - a tool to build web pages according the *ArtSite* data model, the *DataEditor* - a typical content management system and the *DataBaseConstructor* - a tool to create and manage the data base schema (its relations and relationships).

The server consists of the page *parser* and an application server (which is a collection of server side parts of the software modules, and the database manager).

The system also supports the roles, according to the specification, presented in the introduction to this article.

To illustrate the presented ideas we will follow a simple example - a user authorization web page as presented in Fig.3.

The *PageConstructor* was used to create the page as a combination of *blocks*.

The Fig. 4 shows the web page presentation in the website database, as it is seen through the interface. The following color schema is used to distinguish the different types of



Figure 3: A login webpage

blocks:

- blue - a regular *block* - a container which consists of the header and footer (HTML codes stored in the system database);
- dark grey - a *template* - a *block* which consists of the header, the template body and the footer (HTML codes with special tags);
- light grey - a function of type *condition* (a *block* with associated Java code, stored in the database);
- green - a container for the child *blocks*. The child *blocks* are processed, when the *function* of type *condition* returns "true";
- red - a container for the child *blocks*, which are processed when the function of type *condition* returns "false";
- yellow - a function of type *cycle* (a *block* with associated Java code);

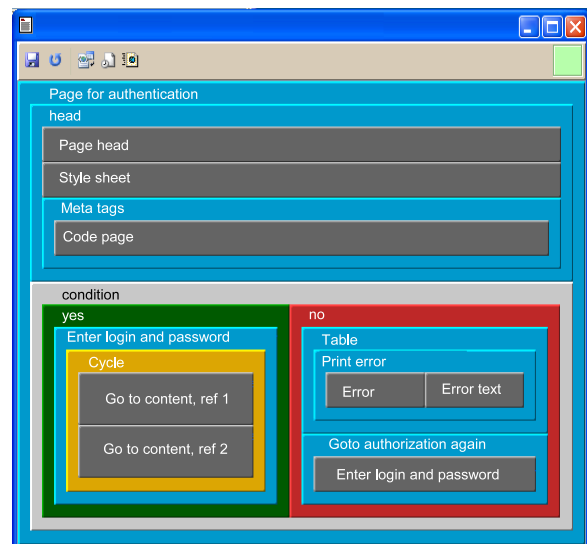


Figure 4: The login webpage structure

Several *blocks* are used to build the page as can be seen in the figure. The *blocks* presented at the top of the figure are used to create the meta tags, the style sheet, and the codepage of the presented web page.

The most interesting *block* is one of type *condition* (light grey in the figure). The bulk part of the Java code, associated with this *block* is presented in Fig.5. This code is stored in the database and can be accessed from the menu associated with the *block*. The code receives the user name and its password as entered by a user, makes the database request to analyze if the presented data is valid, and produces a return code (true or false) depending on the query result. If the result is "true", the child *blocks* from the "green container" will be processed, which generate the success greeting, and display the following content. If the result is "false" the child *blocks* in the "red container, producing the error message, and the request to repeat the authorization procedure, will be used to prepare the final representation of the web page.

As can be seen in the Fig.5, the SQL code in this example is actually inserted into the host Java language. In this sense the example contradicts the *Artsite* data model presented in Fig.1. This contradiction is due to the current implementation of the software package. The independent storage of the SQL queries can allow their usage for different *blocks* which may request the same datasets from the project database, but utilize the different logic using the different *functions* associated with the *blocks*.

A more general example is presented in figures 6 and 7,

```
String login = "";
String password = "";
boolean my_ret = false;

for (int i=0;i<parameters.size();i++){
    Parameter param = (Parameter)parameters.elementAt(i);
    if (param.name.equals("ln")) {
        login = param.value;
    } else if (param.name.equals("pw")){
        password = param.value;
    }
}

if (login.length() > 0) {
    try {
        Statement statement = connection.createStatement();
        String query = new String("select * from cms_users where login='"+login+"' and pass='"+password+"'");
        ResultSet result = statement.executeQuery(query);
        while (result.next()) {

            String sid = getSID();
            g_sid = sid;
            query = new String("insert into cms_sessions (sid,uid,ip,created) values ('"+sid+"','"+result.getInt("uid")+ "','"+ipaddr+",now());");
            int iresult = connection.createStatement().executeUpdate(query);
            if (iresult>0){
                my_ret = true;
            }
        } catch (SQLException e) {
            System.out.println("ERROR "+e.getMessage());
        }
    }
}
return my_ret;
```

Figure 5: The Java code associated with the block

where a webpage constructed by the *PageConstructor* is presented, along with its structure, as it is seen in the *system architect* interface. The color scheme of the *blocks* upon which the page is constructed is the same as in the example discussed above.

4. CONCLUSIONS

The first conclusion which is interesting to present, is that the proposed model can be applied not only to the web pages, but to any kind of reports, which need to be based on the dynamically updated content. For example a special version of the *PageConstructor* can be developed to support the RTF file format, or any other document formats. To support such an algorithm, the data structure presented

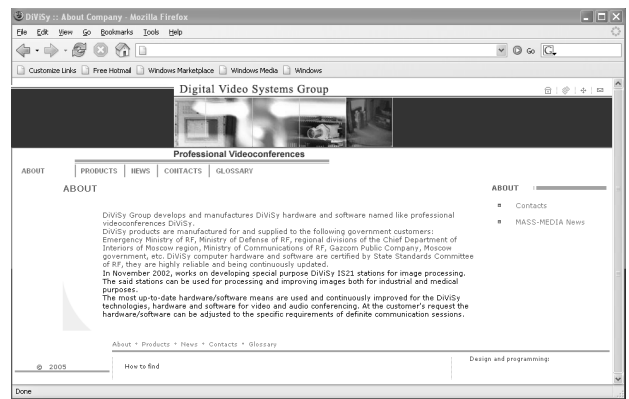


Figure 6: A web page example

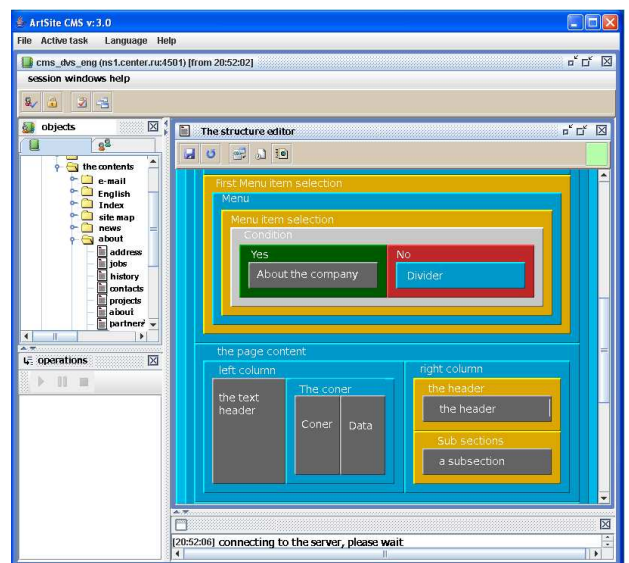


Figure 7: The web page structure, as seen by the system Architects

in Fig. 1 can be modified by adding the relations to support the report types.

The second conclusion is, that the proposed model is consistent with many other ideas of the content management systems, because the *blocks* are thought to be stored and processed upon a distributed database management systems, with many content sources. The proposed model is also consistent with caching systems (such as one proposed in [5]), furthermore it allows to cache only the static parts of the web pages.

One interesting direction of the future work is to develop a conception of the tool, its data model and algorithms, which would use a RDBMS or ORDBMS as a storage for the software components such as classes, methods, and relationships between them. This model may support all of the Object Oriented features of the modern languages, such as Java. If such a model is implemented in the *PageConstructor* described above, the software could become an Integrated Development Environment, supported by the powerful instruments of the RDBMS.

5. REFERENCES

- [1] Louis Weitzman, Sara Elo Dean, Dikran Meliksetian, Kapil Gupta, Nianjun Zhou, Jessica Wu. Transforming the content management process at IBM.com. Conference on Human Factors in Computing Systems Case studies of the CHI2002—AIGA Experience Design FORUM, pages 1-15, Minneapolis Minnesota, 2002.
- [2] Michael Stonebraker, Joseph M. Hellerstein. Content integration for e-business. Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, California, US, Pages 552-560, 2001.
- [3] B. Surjanto, N. Ritter, H. Loeser, L. XML Content Management based on Object-Relational Database Technology. Proceedings of the First International Conference on Web Information Systems Engineering (WISE'00)-Volume 1, page 70, 2000, ISBN:0-7695-0577-5-1.
- [4] Thomas M. Connolly, Carolyn E. Begg. Database Systems. A Practical Approach to Design, Implementation, and Management. Addison Wesley, UK, Fourth Edition 2005, ISBN:0-321-21025-5.
- [5] TK. Selcuk Candan, We-Syan Li, Qiong Luo, Wang-Pin Hsiung, Divyakant Agrawal. Enabling Dynamic Content Caching for Database-Driven Web Sites. ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA.