A hierarchical file system interface to database-based content management application

Ivan Voras, Kristijan Zimmer, Mario Žagar
Faculty of Electrical Engineering & Computing, University of Zagreb,
Unska 3, 10000 Zagreb, Croatia
{ivan.voras, kristijan.zimmer, mario.zagar}@fer.hr

Abstract: *When considering the usability of computer applications one, of the most important factors is the interface they provide for data manipulation. Historically, though each application has defined it's own user interface, some types of interfaces have emerged as most applicable for certain types of data. Over time, some applications (or types of applications) have gained enough popularity that they became ubiquitous and well-known to most computer users to the extent that users consider them the norm, and as such, optimal for their daily tasks. In this paper we present an idea and implemented method of exposing data from a web content management system in the form of hierarchical file system, manageable and editable by usual file management and office application tools.*

Keywords: user interface, web content management system, web application, file system, WebDAV

1. Introduction

This paper describes and discusses implementation of a file system-like interface to data in the FERweb content management system (CMS). "FERweb" is a colloquial name for the web content management system created and maintained at the University of Zagreb, Faculty of Electrical Engineering and Computing during the period of years from 2001. to 2005. The project was started as a research project done by faculty staff and undergraduate students, but has evolved in a self-supporting and viable project, and has as such surpassed its original goals. The project was continued in spirit by the new CMS system named "Quilt."

As the CMS system is used campus-wide, by both the staff and the students, investigation into better user interfaces is always ongoing, in the hope to provide better end easier to use services to all. Basic idea behind the work described in this paper is that a system doesn't have to be homogeneous in operation. With advances in interoperability available in off-the-shelf software products, a system can make use of components that are best suited for its particular purpose. A system like that can utilise a successful and widely used third-party software products and take advantage of the familiarity users have with such software.

2. On applications and user interfaces

Most current graphical user interfaces (i.e. interfaces between computer data and/or programs and the users) are abstractions, and the trend is to increase the level of abstraction with each new generation of the system. In the last decade or more, the prevailing abstraction for common types of data used in common desktop environments is to represent them as "documents." At the same time, the distinction between what is application and what are data is increasingly and intentionally being blurred. The discussion about reasons and correctness of this trend is beyond the scope of this document, but rather we wish to concede that this state is dominant and to make as much use of it as possible.

One consequence of the described trend is that casual users (especially ones not involved in the technological aspects of computer operation) do not perceive the existence of applications as such, but are content to deal with their data on the highest possible level, in the way they have been used to or taught to. In addition, the visual aspects of the methods used in presentation and accessing the documents greatly influence the users' perception of the documents themselves [1][2].

As an example, users tend not to think of their textual documents as "a stream of bytes generated by the word processing application" (which is the low-level technical interpretation),

or even not as "a file located in the file system that is labeled as a textual file" (the "middle level" of abstraction), but as "that graphical widget which looks like a piece of paper with a blue letter W written on it". This also relates to the gradual progress of using abstractions through time and can be compared with computer programming languages: early computer users were regularly experts and as such had no problems with low levels of abstraction that exposes technical details. Starting with around 1980-ies computers have became more ubiquitous and more accessible to wider range of users and the predominant level of abstraction for documents and applications has became "the file system". Currently the level of abstraction is being raised once more in a way that users are not aware of how the underlying system works or which application handles which document.
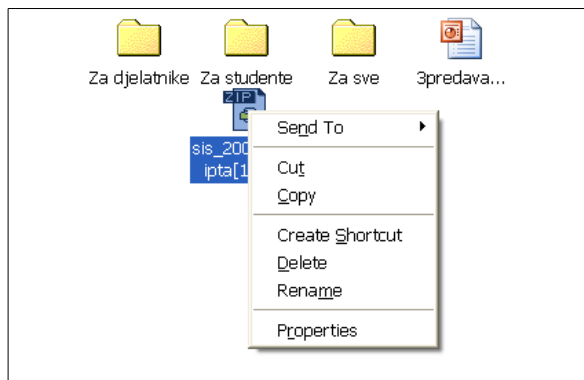

Figure 1. Files represented as visual objects in a graphical user interface

As the predominant office application suite today is Microsoft Office™, it's interface is used as a "golden standard" which other applications tend to mimic. Since the global market share saturation of Microsoft Word™ is up to 95 percent [3], most of current users are not even aware of other solutions and expect the Microsoft Office suite to be present on all their workstations.

2.1. Leveraging existing applications

Some of the ways existing applications can be used from third party systems to provide a well-known environment for creating and editing content are:

- Manually, by which is meant that the user is responsible for uploading content

created by a desktop application into some other system
- Through an interface or plug-in architecture for embedding software components into client applications such as the ActiveX and COM (Component Object Model)
- Through redirecting file system operations such as loading and saving user's data

In this work, a method and implementation is presented that uses the method of redirecting file system operations (of standard office application suites) to store and retrieve user-created content to and from a web content management system. This is accomplished using the WebDAV standard network protocol for authoring and distribution of files.

3. About WebDAV

The name "WebDAV" stands for "Web Distributed Authoring and Versioning." It is a standard governed by IETF working group and is a network protocol for sharing and distribution of files and file-like resources over computer networks. The basic protocol was formalised in a Request for comments (RFC) document RFC2518[4], but the development of additional features is ongoing.

The WebDAV is a textual network protocol implemented as a set of additions to the HTTP version 1.1[5]. In particular, any WebDAV-compliant server is also HTTP/1.1-compliant. This level of backwards compatibility allows using ordinary HTTP/1.1 clients for simple retrieving of resources. The protocol uses XML to transfer additional information related to the resources managed such as creation time, resource type, author, etc. Information such as that is commonly called *metadata*.

3.1. Overview of the WebDAV protocol

WebDAV protocol extends HTTP with additional methods (commands):

- PROPFIND – used to fetch metadata information about resources
- PROPPATCH – used to update resource metadata
- COPY – used to copy resources between collections remotely on the server
- MOVE – used to move resources between collections remotely on the server

- DELETE – used to remove resources
- LOCK – used to acquire locks for implementing contention control
- UNLOCK – used to release locks acquired by the LOCK method

Standard HTTP methods OPTIONS, GET, HEAD and PUT are extended to support additional semantics. The POST method is left undefined.

Several HTTP headers are defined to extend functionality by providing announcements for server capabilities, use and manage resource locking and additional parameters to other methods.

A typical example of a WebDAV request is presented in Fig. 2.

```
PROPFIND  /folder/ HTTP/1.1
Host: www.foo.bar
Depth: 1
Content-Type: text/xml; charset="utf-8"
Content-Length: 111

<?xml version="1.0" encoding="utf-8" ?>
 <D:propfind xmlns:D="DAV:">
   <D:allprop/>
 </D:propfind>
```

Figure 2. WebDAV protocol example

The relatively simple and straightforward way in which the WebDAV protocol is implemented has been beneficial for its spreading and usage. As of time of writing of this paper, all major web development and office application suites natively support at least the basic WebDAV features.

4. Data representation and storage in the FERweb CMS system

As a direct consequence of its purpose as a web content management system, all data in the FERweb system are grouped into "pages." Each page is referenced by a unique name (URL part) and can contain an arbitrary number of arbitrary "modules." Each module is responsible for a certain type of functionality in the system – for example: content articles, news articles, user forums, file repository, etc. The system supports an arbitrary number of users and allows for creating a complex set of roles and privileges that restrict users' abilities to manipulate data.

All CMS data are stored in relational SQL database (PostgreSQL). In the database there is an implicit distinction between tables belonging to the system core (e.g. tables that record information about pages, users and access rights) and tables used by CMS modules to store their data. Each piece of data generated and handled by a module is ultimately linked to a single web page (though some modules may have the ability to duplicate and/or create cross-links to certain data) but each module can hold an arbitrary number of data pieces on a single page.

5. Adapting CMS data into hierarchical file system-like structure

As each page in the system is identified by a unique URL-like resource name (e.g. "/referada/obavijesti"), it is natural to make use of this information to build a virtual hierarchical directory structure. Additional issues needed to be resolved were:

- Presentation of page-related metadata such as page title, keywords, description, layout of modules and access permissions
- Presentation of modules and modules' data

The first issue was resolved by presenting such metadata as ordinary text files inside the virtual directory structure. Such files are generated on the fly when they are referenced. When saved, their contents are parsed and appropriate data in system tables are updated. The contents of these files are virtual in the sense that they are never stored; their syntax is intentionally similar to well-known system configuration files (one example of which are ".INI" files).

Issue of presentation of modules and their data inside the user interface could have been handled in several ways, most notable of which are:

- Presenting modules' contents directly as files in the virtual directory structure, with additional metadata configuration files dispersed among them
- Presenting each module as a virtual subdirectory containing files pertinent to that module only, with metadata configuration files placed in module-specific locations

Considerable thought was given to both methods, and the latter approach was considered to be better because the former has the potential to result in a crowded directory with lots of unrelated files. In addition, the latter approach has the benefit of integrating neatly with the concept of modules on a web page and is intuitive to existing users of the system. To distinguish modules from pages that are deeper in the file system hierarchy, virtual directories that represent them have a "$" character prepended to their names.

Each module takes on the responsibility of generating its own file and subdirectory structure in which records from database are represented by files of appropriate types. For example, module that handles and displays textual data can expose the content as .html files, and module that presents a web calendar functionality can expose the data as virtual files containing data in iCalendar format.

The end result is a hierarchical file system where individual pages are presented in a tree of directories, each of which contains metadata configuration files and additional subdirectories representing individual modules. An example of such structure is given in Fig. 3.
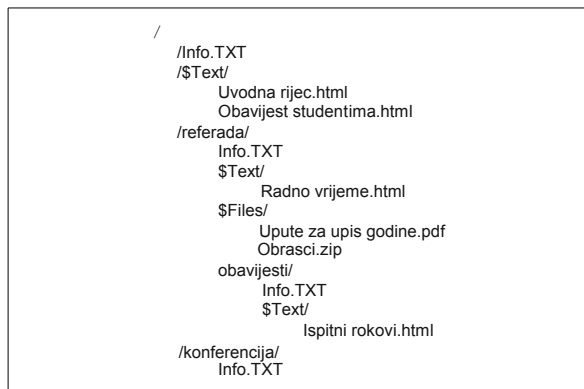
```
/
    /Info.TXT
    /$Text/
        Uvodna rijec.html
        Obavijest studentima.html
    /referada/
        Info.TXT
        $Text/
            Radno vrijeme.html
        $Files/
            Upute za upis godine.pdf
            Obrasci.zip
        obavijesti/
            Info.TXT
            $Text/
                Ispitni rokovi.html
    /konferencija/
        Info.TXT
```

Figure 3. Example of generated virtual hierarchical file system structure

In Fig. 3 the exemplary structure was generated for four web pages from the CMS: "/" (the root page), "/referada", "/referada/obavijesti" and "/konferencija". The root page has two text elements handled by the "Text" module, the "/referada" page has one text element and two files handled by the "Files" module, the "/referada/obavijesti" has one text element, and "/konferencija" is empty of modules. Names of files are generated from

records stored in the database, in particular, from "title" or "description" fields of appropriate tables.

5.1. Implementation details

The working implementation of concepts described previously is made as a standalone WebDAV server application written in Python programming language. Basic architecture of the application is comprised of three layers:

- CMS module interface
- CMS system services
- Protocol (network) interface

With possibilities offered by object-oriented design, each layer was made to be separable and replaceable.

CMS system services and module interface are directly tied with data representation in the database. Their purpose is to adapt data between data records in the CMS database and a hierarchical file system. CMS module interface consists of a set of OOP classes, each of which handles data for one particular CMS module. These classes are also called "module adapter classes." The system services layer is responsible for common operations such as authenticating users, enforcing users' privileges and mapping URL-identifying data from client requests into a hierarchical directory tree. Finally, the protocol interface is responsible for translation of internal data structures into protocol-specified format and exposing them in that form over the network interface.

Each client request begins and ends in the protocol interface layer. After receiving and interpreting a request, this layer either responds to it by itself (in case the requests is simple and concerned with details of the protocol), or passes it to the CMS system services layer. The next layer again checks if it can satisfy the request by itself (for example, if it's a request for a directory listing or concerns security validation) and does so if it can. If the request is determined to be made to a module, an object of appropriate module class is created and the request is passed to it. Module and system interface layers respond to requests by generating response objects which are received by the protocol layer and converted into specific protocol data. Each layer can make a decision (based on additional information available at each layer) that the request is invalid

or malformed, or that an error condition occurred, and respond by returning appropriate data to the calling layer.

This software architecture is designed to be extensible. New CMS modules can be supported by writing an appropriate adapter class. Since all network protocol handling is also isolated, it can be extended to support other protocol such as the File transfer protocol (FTP) and network file sharing protocols.

6. Usage and usability

Support for the WebDAV protocol is present in file management shells of all major operating systems, including Microsoft Windows 2000 and up, MacOS X, GNOME and KDE environments for Unix-like operating systems (including Linux). In those systems WebDAV resources are for the most part indistinguishable from local files and directories.

Mimicking a standard hierarchical file system opens new grounds for data manipulation that were difficult or impossible to implement before, such as changing the structure of a web site by visual drag-and-drop method and editing large and complex documents. Figures 4 and 5 show a document being edited in Microsoft Word using its HTML-editing capabilities and then "published" to the server by using the standard *Save File* operation. The WebDAV server is responsible for loading data from the database when the "document" is accessed, and saving it to the database when the document is written.

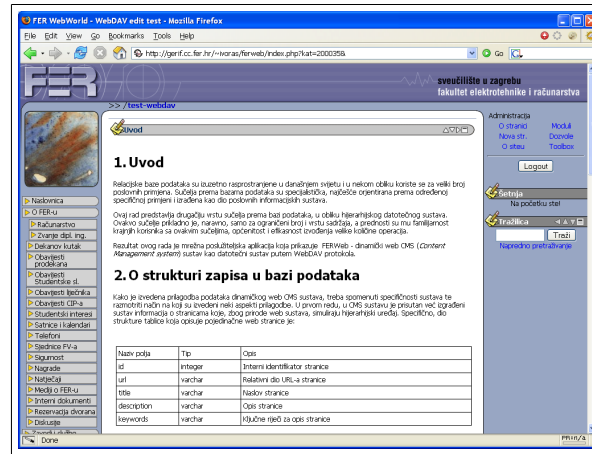**Figure 4. A HTML resource being edited in Microsoft Word**

Figure 5. HTML resource from Fig. 4 accessed through the web CMS system

In an internal study conducted with existing users of the CMS system it has been found that almost all users find this method of creating and editing content attractive and, in case of large documents, easier because of familiar tools (such as Microsoft Word) and advanced capabilities those tools offer as compared to those available through a web-only interface.

7. Conclusion

Training users to use new tools can be long and expensive, and thus it's important to leverage users' familiarity with their usual tools to maximum extent. This paper has presented a method and an implementation of representing data from a web content management system (the FERweb system) as a consistent hierarchical file system usable on remote network clients via the Web Distributed Authoring and Versioning (WebDAV) protocol. In such environment, various database records are presented as separate entities (nodes) in a virtual file system.

A standalone application was created to act as a WebDAV server. In the application there is a clear distinction between network protocol layer, the CMS services layer and the CMS module interface layer. Each CMS module, responsible for one type of data and functionality in the CMS, can have corresponding "adapter" code inside the CMS module interface layer (implemented as an application object) that transforms CMS data from the database into arbitrary file system-like data. The issue of
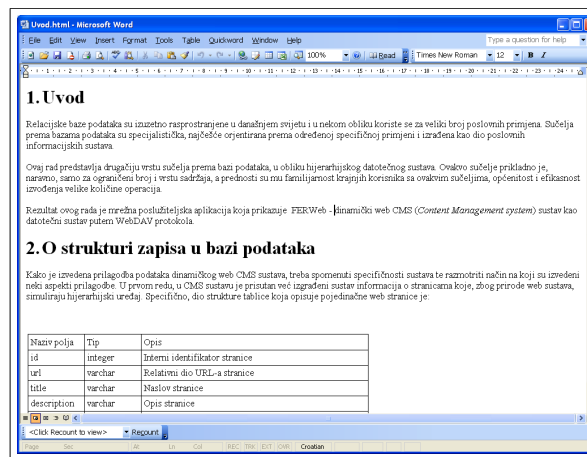
metadata was solved by creating virtual text configuration files editable by text processing applications, the contents of which is parsed when the files are written to the file system. Such file system can be accessed and manipulated by most current operating systems and content-creating applications.

Responses from our end users have shown that ideas and concepts presented here are received warmly and appreciated.

8. References

[1] Apple Computers: Apple Human Interface Guidelines, online publication, http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/

[2] Microsoft: Official Guidelines for User Interface Developers and Designers, online publication, http://msdn.microsoft.com/library/en-us/dnwue/html/welcome.asp

[3] Michael J. Martinez: Innovation needed before techs can grow, USA Today 2004-10-10

[4] Y. Goland et al, HTTP Extensions for Distributed Authoring – WEBDAV. IETF RFC Document #2518, 1999.

[5] R. Fielding et al, Hypertext Transfer Protocol – HTTP/1.1, IETF RFC Document #2616, 1999.

[6] F. Dawson and D. Stenerson: Internet Calendaring and Scheduling Core Object Specification (iCalendar), IETF RFC Document #2445, 1998.