ABSTRACT
In this work we present techniques and tools that enable effective reverse engineering procedures for web applications that were developed using the promising ASP.NET technology. We deal with model-driven development in its reverse aspect by implementing reverse engineering methods. Our implemented methods model web applications using a well-known, web oriented and robust language, namely WebML. This is, to the authors’ best knowledge, a novel re-engineering transformation. In this paper we propose a method to reverse engineer web applications in order to extract their conceptual model using WebML notation. Moreover, we present an efficient tool we have developed in order to implement the proposed method, along with a study of the application of our tool to an exemplar, content-management web application. The overall results are quite encouraging and indicate that our approach is efficient.

Categories and Subject Descriptors
D.2.2 [Programming Languages]: Design Tools and Techniques.
D.2.7 [Programming Languages]: Distribution, Maintenance, and Enhancement - Restructuring, reverse engineering, and reengineering.

General Terms
Design, Management, Standardization.

Keywords
WebML, Reverse Engineering, Web Applications, ASP.NET

1 INTRODUCTION
Novel World Wide Web applications are mainly developed in very short time and moreover the legacy applications are being modified quickly and frequently due to market needs. Due to the above circumstances during development and maintenance, programmers do not always take software engineering principles into account, neither do they follow well-defined, documented software development methodologies. The lack of a well structured method during development and maintenance of web applications is considered the main reason for the existence of software with haphazard architecture and inadequate or inexistent documentation. This software can be analyzed/modified with considerable effort and time [8].

Reverse engineering methodologies in the field of traditional software systems have been proved very useful to support maintainability of such systems. These methodologies provide approaches in order to retrieve information and to provide documentation to software systems that are little or not at all documented. The problem of defining similar techniques, approaches and tools to web development is an open issue to the research community. According to Chikofsky and Cross [5], reverse engineering is: “the process of analyzing a subject system to: a) identify the system’s components and their interrelationships and b) create representations of the system in another form or at a higher level of abstraction”

A reverse engineering process can be used in order to achieve several goals such as: Development of an application, Re-documentation of a legacy application, Retrieval of its architecture, Application testing, Application maintenance, and Quality evaluation.

Under this viewpoint, the proposed reverse engineering approach is an important step towards implementing a tool to represent an application at a higher level of abstraction that is, in terms of a composition of standard patterns. The rationale behind this approach lies in the transformation of a web application into a WebML model by means of an intermediary tree (or in fact a directed acyclic graph – DAG data structure), which is compatible with the model-clone recognition approach presented by some of the authors in [15]. The approach in [15], when applied to an application’s WebML conceptual schema, results in effective design solutions, as it facilitates reuse and consistency in the development and maintenance process. As a next step, we present in this work a method that allows us to transform existing web applications developed with ASP.NET, into WebML models represented first with the appropriate graph -DAG- to enable further meta-modeling. Overall, WebML [3] was chosen for the conceptual representation since it is a robust, omnipotent modeling language and it is supported by WebRatio CASE tool.

In the rest of the paper we are first presenting existing work in the area of reverse engineering. Consequently, we propose a method to reverse engineer web applications in order to model them with
the use of WebML. Moreover, we present the tool we have developed in order to implement the proposed method, along with a study of the application of our tool to an exemplar, content-management web application.

2 PREVIOUS WORK

Analysis of legacy web applications has been the subject of many existing works. The main bulk of those works transforms existing software engineering tools in order to make them suitable to operate in a web context or propose ad hoc methods to reverse engineer applications.

In this vein, Hassan and Holt [11] describe the transformations they have made to Portable Bookshelf Environment (PBE), to acquire the architecture of a web application. Moreover, Schwabe et al. [9] define an operational framework for design reuse in web applications. They separate issues regarding the behavior of the application from navigational modeling and interface design. Boldyreff et al. [2] propose a system that exploits the traditional techniques of reverse engineering to extract duplicate content and style from pages, so as to restructure them and to improve their maintainability. Cloning issues in web applications have also been investigated in the work of Jarzabek and Rajapakse [13], which have shown that cloning can be as high as about 40% in web applications. They also describe methodologies to infer clones and the respective similarity metrics.

Other tools that serve reverse engineering purposes include VAQUISTA [4] and WARE [1, 8]. The method in [1] defines all the necessary tasks that have to be carried out in order to acquire a set of views from the web applications, at different abstraction levels. Paganeli et al. [14] describe the TERESA tool, which, by means of source code statistical analysis, produces a task-oriented model of a web application, where each task represents isolated functions of a single page that were generated after user request. The produced model is suitable for usability evaluation of a web site, or for the detection and registration of user-profiles in this website.

Other approaches cope with the analysis of web applications aiming to assess or improve quality of those applications. An analysis approach that allows the test model of a Web application to be retrieved from its code and the functional testing activity to be carried out is proposed in [6].

Finally, Girardi et al. [10] propose techniques and algorithms that support restructuring of multilingual applications. Their approach can be used to produce maintainable and consistent multilingual applications.

3 THE REVERSE ENGINEERING PROCEDURE

In this section, we present the main concepts of our reverse engineering method. We apply this method to web applications that have been created with use of ASP.NET, in order to model them with WebML. The advantage of the proposed method is that it can also be applied for reverse modeling applications with use of other modeling languages with a few straight-forward alterations.

In the following, a detailed description is included concerning the main and particular characteristics that were taken into consideration for reverse engineering and modeling of ASP.NET applications. An ASP.NET web application consists of one or more ASP.NET pages or web forms, files with business intelligence (code behind files), configuration files and XML meta-data description files.

A web form is stored in aspx files, which are basically html-like files that additionally support the containment of certain .NET specific labels (tags). In the aspx file, the layout and page appearance are determined and usually it is inter-connected to a code-behind file, which contains the application logic for the objects (web and other components) included in the aspx file. The code-behind files can include handlers for various events, utility methods, element statements – objects of aspx page, as well as certain interrelations to data. A label/tag or directive in the beginning of an aspx page determines the name and the locality of the corresponding code file. For example suppose that we have an aspx page with the name WebForm1.aspx. This aspx file includes the following directive:

```csharp
%@ Page language="cs" Codebehind="WebForm1.aspx.cs" %>
```

This directive informs us about:

- the file name (here WebForm1.aspx.cs) that contains the code of the corresponding Web form (here WebForm1.aspx) and
- the language which the code has been written in. In particular, from the previous directive it appears that C# has been used for code development. Other popular languages that are supported are Microsoft Visual Basic .NET and Microsoft Jscript .NET as well as versions of Python and others.

A basic trait of ASP.NET is the separation of Web forms that contain the presentation logic, from the code-behind files in which the operational logic is contained (i.e. business logic). This separation allows separate analysis and treatment of code files from object files, which is obvious to the user.

3.1 Methodological Approach

The proposed approach is based on the analysis of application source code. For the static code analysis the creation and utilization of two analysis parsers is employed. The first parser goes through the aspx files and creates a tree for each page with html and aspx tags. Then, with use of a correspondence matrix, we match labels that interest us with the most appropriate WebML units. This way, we create a first estimation of the units that are contained in the pages at hand. Finally, the second parser goes through the code-behind files. The main objective of this second analysis is to locate the data sources that each unit, located during the first analysis of the aspx pages, utilizes to draw information.

The proposed methodology steps are presented in Figure 2. The steps of the method are presented in detail in the next paragraphs.

3.2 Creating ASP.NET tag trees

3.2.1 Pre-processing

In this step, aspx documents are initially analyzed. Each aspx page consists of simple informative text and labels (tags). Every tag comprises a name and an optional list of predicates and it is found between a pair of the open "<" and close tag symbol ">

A
sequence of characters that is found between two labels is considered to be text. The labels are usually presented in pairs, apart from certain language specific exceptions. Tags that do not begin with a forward slash ("/") are called "starting tags", otherwise they are named "ending tags". An "ending tag" consists of a "forward slash" and the name of a corresponding "starting tag".

Figure 1 Steps of Methodological Approach

In order to successfully analyze an aspx page, it should be well structured. The term "well structured" implies that this page should satisfy the following conditions:

- Equality and inequality symbols when needed should be represented using "&lt " and "&gt ", respectively. No orphan standard (in)equality symbols appear (i.e.: "<", ">").
- All tags are presented "in pairs", apart from tags <br>, <img>, <hr> etc., which are reported in the specifications of XHTML and ASPX.
- Quotes are used to include values of tag predicates, (e.g. <a href="www.acm.org">).
- No cross-nested tag pairs appear in the page. The tags should be included with no cross-overlapping. For example the piece of code "<a> ...<b> ...</a> ...<b> ...</b>" is not correctly structured. The above piece of code would be analyzed, only if it were in the form of "<a> ...<b> ...</b> ...</a> ...</a>".

Pages that do not have this format should be changed and replaced by "well-structured" ones. This transformation, which is also known as page normalization, includes:
- Absent or incomplete tag formulation (Import of labels that are absent).
- Tag restructuring.

The page normalization is essential, in order to enable tree-based representation so that each node will have only a single direct ancestor. This can be achieved using tools such as HTML TIDY [12] which is usually included in the standard restructuring intelligence of visual web development tools (e.g Visual Studio .Net 2003).

3.2.2 Tree Representation of Pages

A well structured page can be directly represented using an ASP.NET-tag tree. The following steps lead to the creation of such a tree:

- A node is created when meeting the starting tag of tag pairs. The node name is derived from the tag name. Usually, in a well structured page, the first node, that is created, contains the text <html>.
- If the "starting tag" contains predicates, the corresponding leaf-children, for each predicate, are created. The leaves contain the information and the values of the predicates.
- When a nested "starting tag" is found, a new tree node is created. The new node is child of node that represents the already encountered "starting tag". We apply the previous step to create the appropriate leaves of the new node, based on the list of predicates of the tag that was met.
- When an "ending tag" is met, no node is created.

For example, suppose that we have an aspx page, the code of which is presented in Figure 2. The corresponding tag tree appears in Figure 3.

3.3 Matching ASP.NET controls to WebML

In this step, we distinguish the nodes or the leaves of our interest. This is achieved with the use of correspondence table that is presented in the following.

For each page we go through the nodes of its representing tree, created in the previous step. If a node-tag belongs in the table, we create an entry in a text file. The entry contains the appropriate type unit which corresponds to the tag and the object id that represents the node. We consider that each object in an aspx page has a unique id. Detection of the object id is achieved by running through the leaves, which represent the list of predicates. The
mapping of various html and ASP controls into WebML units which is presented in Table 1 is simple and straightforward.

The units that are presented after the completion of this step are not the final ones, as they can change, after the implementation of data source exporter described in following.

Table 1 Matching html-asp tags and WebML Units

<table>
<thead>
<tr>
<th>HTML Controls</th>
<th>WebML</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;a href=&quot;…&quot;&gt;</td>
<td>Link</td>
</tr>
<tr>
<td>&lt;input type=&quot;text&quot;&gt;</td>
<td>EntryUnit</td>
</tr>
<tr>
<td>textarea&gt;</td>
<td>EntryUnit</td>
</tr>
<tr>
<td>ASPX Controls</td>
<td>WebML</td>
</tr>
<tr>
<td><a href="">asp:DropDownList</a></td>
<td>IndexUnit</td>
</tr>
<tr>
<td><a href="">asp:ListBox</a></td>
<td>IndexUnit</td>
</tr>
<tr>
<td><a href="">asp:DataList</a></td>
<td>MultiDataUnit</td>
</tr>
<tr>
<td><a href="">asp:DataGrid</a></td>
<td>IndexUnit</td>
</tr>
</tbody>
</table>

3.4 Database Extractor

At this step we process the text files created during the previous step and analyze the source files connected to .aspx pages that correspond to the text files.

Each text file, as mentioned, includes tag ids appearing in .aspx pages. For each id we parse the corresponding text file in order to locate database calls. More specifically, we seek SQL keywords, such as SELECT and UPDATE and then words like FROM and WHERE in order to determine from which database table the specific unit retrieves information. It is quite possible that for certain units we will not be able to fulfill this task either because we need to execute a stored procedure or because the query string belongs to another page or is embedded in a DLL.

If the search is successful and we manage to find the SQL query string corresponding to some unit, we have the option to change the unit type. Consider for example that we have mapped an <asp:datagrid> tag to an index unit and that during the analysis of text files we have found that the term “WHERE” of the corresponding SQL query string, is used for searching objects with the primary key of a table. Hence, the returned object is obviously unique. Therefore, the specific <asp:datagrid> tag must be represented as a data unit and not as an index unit.

Having processed all the retrieved query strings, we can extract useful information for each unit, e.g. which table it uses, which fields-columns will appear as well as if the order of the results is ascending or descending. The results are again stored in the same text files. At this stage, the programmer’s or database designer’s intervention is frequently required.

3.5 Representing the application with a tree

During this step we traverse the application directories and create a tree according to the folders contained in the application, the pages and the units of each page.

The text of the tree root will get the name of the application. Next for each .aspx file contained in the initial application folder, children nodes of the root are added. We iteratively add children nodes for each subfolder and their corresponding .aspx files. Next, we add to the node for each .aspx file, new children nodes. The text of the latter is the type of the recognized units along with the unit id. We finally add leaves to the nodes that represent units.

The text of the leaves contains the various attributes we have extracted for each unit, in the previous step.

3.6 Adding links to the tree

After having created the tree, the created text files are checked again, to find out whether they contain a link that either carries information (contextual) from a unit to a page or another unit, or simply leads to another page without carrying information (non-contextual). In case we find a link, we add edges between the nodes connected by the link.

The search procedure is carried out after we have already formed an initial application tree, so that the units pointed by links truly exist. Therefore, if a unit pointed by a link does not exist, we assume that we have found a wrong link. At this point, the application programmer is notified to either delete the link from the specific page or correct the mistake. Ultimately, after all corrections have been applied, the tree takes its final shape.

4 METHODOLOGY LIMITATIONS

It is obvious from the above steps of the proposed method that we might encounter problems with computer resources and memory usage in the case of applications that contain a large number of files, since we go through each file several times.

Furthermore, in order to perform static code analysis of a web application, the programmer is required to have all the pages and the files of the application at his disposal. However this is not always possible, since web applications are developed using many different languages (e.g. C#, VB.Net, JScript) and contain components and objects whose source code may not exist. During the development of an application the programmers may have purchased an object in a binary code form, thus they do not have the source code. It is also possible that the application programmers do not want to release some files of the applications due to sensitive data they may contain. It is easily understood according to this observation that: a) code analysis of a web application might be a complicated procedure, b) the right conclusions are not always reached and c) the intervention of the programmer or developer is frequently required.

Finally, the proposed method can retrieve and recognize only those units that have been identified in the mapping table. Thus, intervention is required to define siteviews and areas.

5 DESCRIPTION OF THE TOOL

For the purposes of this paper, a tool was created that receives as input the path and the database schema of the application we analyze, and visualizes the results in a tree form. In the following we present selected snapshots of the implemented analysis tool.

After having chosen an existing ASP.NET application, the aspx pages are analyzed and a form appears with the names of the files created on the left of the form (Figure 4). By choosing a text file, the user can see its contents, i.e. the identified units and links, on the right part under the title “Contents”. Next, the user must press the “Database extractor” button to continue the process.

By pressing the button, the files are analyzed and a new form that contains the new text files of the application is presented to the user (Figure 5). The user must press the “Data Visualization” button to display the tree representing the application.
AN EXEMPLIFYING PARADIGM

For test purposes of the developed tool we used an example application for news management. Such applications are met in many websites, e.g. the popular national portal http://www.flash.gr

The tables in Figure 6 are the database tables, which were created during the application development. More specifically, the “Categories” table includes the various news categories, visible to our application and the “Articles” table contains the news articles.

Next we present two of the created pages. In the first one (Figure 7), the user is presented on the left with a list of news categories and on the right some excerpts of news-articles, corresponding to the category he has chosen. In order to see the full article, the user selects the title of the article he wants to see. Hence, a new page appears (Figure 8) containing on the right side the full article text and on the left side the news categories.

The tree that was created by applying the proposed method to the exemplar application is shown in Figure 9. Red arrows represent hyperlinks from a unit or page to another page. In Figure 9 we see that Article_page.aspx contains a dataunit “Article” and an index unit “News_Categories”. The News.aspx contains an index unit named News_Categories MultiDataUnit named “Articles”.

On the left part of Figure 10, we present the WebML part of the developed application that corresponds to the tree of the example-application and that was extracted from our tool. The right part presents the application schema, the way it was designed and modeled before the development stage.

By examining the two parts of Figure 10, we can observe that apart from the direction and the type of links, the created WebML schema is similar to the initial WebML schema. Thus, we have managed to successfully model a web application by statically analyzing its code.

Our key aim was to further enable model-clone extraction, maintenance and metric-based evaluation with the further use of our method presented in [15]. As a result we have parsed and transformed the application into a model tree compatible for model-clone extraction.
The presented approach only statistically analyzes the web application code and does not incorporate all metrics of [15], which is a direction we already work towards. Further work includes wider experimentation and fine-tuning of the method for more .NET languages.

![Figure 10 Analysis output (left) & initial WebML schema (right)](image)

### 7 REFERENCES


