Automated Analysis and Evaluation of Web Applications Design: The CMS-based Web Applications Case Study

Vassiliki Gkantouna[1], Athanasios Tsakalidis[1] and Giannis Tzimas[2]
[1]*Department of Computer Engineering and Informatics, University of Patras, Rio Patras, 26500, Greece*
[2]*Department of Computer and Informatics Engineering, Technological Educational Institute of Western Greece, M. Alexandrou 1, Koukouli Patras 26334, Greece*

Keywords: Design Reuse, Design Pattern, Design Evaluation, CMS, WebML, Web Application, Web Mining.

Abstract: This paper addresses the automated design quality evaluation of Web applications built on a CMS platform by inspecting their conceptual model under the viewpoint of consistent design reuse. We have utilized WebML as the design platform of the proposed methodology and we attempt to capture design reuse by detecting all the recurrent patterns within the WebML hypertext model of an application. A pattern consists of a core specification, i.e., an invariant composition of WebML elements that characterizes the pattern and by a number of pattern variants which extend the core specification with all the valid modalities in which the pattern composition can start (starting variants) or terminate (termination variants). We have developed a methodology that automatically extracts the hypertext model of a web application which is subsequently submitted to a pattern-based analysis in order to identify the occurrences of all the incorporated recurrent patterns implying design reuse. Then, we calculate evaluation metrics revealing whether the identified patterns variants are used consistently throughout the application. By using the methodology, designers can detect either effective reusable design solutions consistently used throughout the application model for obtaining certain functionality within the application's context or recurrent design constructs causing design inconsistencies and lowering the quality of the final application.

1 INTRODUCTION

Nowadays, the ever increasing complexity of modern web applications has led to serious problems of usability and has raised the need for new methods enhancing the quality of both the web development process and the final application.

In response to this need, a plethora of Model-Driven Web Engineering approaches (Aragón et al, 2013) has been proposed in the literature along with a number of web design patterns (Bernstein, 1998). They rely on conceptual models and provide developers with formal techniques for an effective design and development process. In an effort to promote design reuse, design patterns support developers with proven solutions that can be reused in different contexts where the correspondence problem arise. This way, reuse based on design patterns allows facing the complexity of Web application development and improving the development process. At the same time, the adoption of design patterns can also increase the application's quality since the use of successful solutions implies a major reliability of the final application. In addition, if design patterns are used consistently in the design of an application, they enhance its usability, on the grounds that it is easier for users to identify reliable expectations about the application's structure and behavior.

Despite the fact that there are catalogues of design patterns ready for use (Welie, 2008; Website patterns, 2015), developers have found it difficult to properly use them. The main reason is due to the fact that there is only a very limited number of experienced web designers who can distinguish which is the appropriate design pattern for solving effectively a particular instantiation of a design problem (Díaz et al, 2009). Another reason is the difficulties that developers encounter to apply the patterns consistently throughout the design model of an application. Especially when they try to apply past experiences and reuse a previously successful design pattern, it is common, due to lack of time, that they do not properly adapt it to the requirements of the new

Gkantouna, V., Tsakalidis, A. and Tzimas, G.
Automated Analysis and Evaluation of Web Applications Design: The CMS-based Web Applications Case Study.
In *Proceedings of the 12th International Conference on Web Information Systems and Technologies (WEBIST 2016) - Volume 1*, pages 130-139
ISBN: 978-989-758-186-1

project at hand, resulting in pattern variants which cause serious design inconsistencies. All the above highlight the need for tools supporting developers inspect and evaluate the consistency of web applications design, even at the conceptual level, in order to discover potential design problems at the early stages of development and facilitate the fast recovery with the less possible effort.

This paper therefore addresses the automated analysis and evaluation of web applications design by inspecting their conceptual model under the viewpoint of consistent design reuse. At the conceptual level, we attempt to capture reuse by detecting the recurrent patterns lying within the model of an application. We have utilized WebML (Ceri et al, 2000) as the design platform of our methodology mainly due to the fact that it supports a concrete framework for the formal definition of data-intensive web applications. We consider that a pattern consists of a core specification, i.e., an invariant composition of WebML elements that characterizes the pattern and by a number of pattern variants which extend the core specification with all the valid modalities in which the pattern composition can start (starting variants) or terminate (termination variants). We have developed a methodology that automatically extracts the hypertext model of a web application and subsequently performs a pattern-based analysis of the model in order to identify the occurrences of all the incorporated recurrent patterns implying design reuse. Then, we calculate evaluation metrics revealing whether the identified patterns are used consistently throughout the application design.

In order to automate the process of evaluating the design of a Web application, we have narrowed down the methodology's scope to the domain of CMS-based Web applications. The main reason for this is that CMSs (Content Management Systems) provide a common base of source code which can be systematically processed for obtaining the automated extraction of the application's hypertext model. The proposed methodology is accompanied by a tool support available in (CMS Modeling, 2015), allowing developers to apply the methodology on websites developed by using the Joomla! (Joomla! CMS, 2015) and Drupal (Drupal CMS, 2015) CMS platforms. Due to space limitations, in this work we present the methodology for the case of Joomla!.

The remaining of this paper is organized as follows: Section 2 provides an overview of the related work and discusses the contibution of this work. Section 3 describes a brief overview of WebML. Section 4 presents in detail the proposed methodology, while section 5 discusses conclusions and future work.

2 RELATED WORK AND CONTRIBUTION

Our primary goal is to address the design quality of web applications by focusing on design reuse within their conceptual model. We consider two main types of reuse: (i) the one which is due to the application of well-known design patterns empirically devised by experienced web designers and (ii) the other which has emerged as a result of the design decisions made by developers for adopting specific reusable design structures to meet the particular needs of an application. The latter type can result in either effective reusable solutions consistently used into the application model for implementing a certain task, or in problematic cases of reuse causing inconsistencies and lowering the application's usability and quality.

To the best of our knowledge, the proposed approach has no counterpart in the field of conceptual modeling. Only the work in (Rigou et al, 2006) can be considered to have a similar point of view, as it examines both of the aforementioned types of reuse in the conceptual schemas of web applications. The authors propose a methodology for detecting and evaluating model clones implying design reuse in the application model. However, this methodology cannot be applied to any web application, since it does not support the automated creation of their model, which is a key point of the methodology.

The other studies in the literature when referring to design reuse in the conceptual model of an application, they consider merely the first type of reuse. These studies are divided into two main categories: (i) the first one focuses on patterns specification after analyzing and reviewing a large number of successful web applications and (ii) the second one focuses on the detection and evaluation of the instances of predefined design patterns within the conceptual model of an application. Regarding the first category, in (Ivory, 2005) the author has examined the characteristics of the design of highly rated websites and present a study about the evolution of web design patterns. In (Fraternali et al, 2008) authors address the design of community-based Web applications by proposing a set of WebML design patterns, identified by reviewing a number of top-rank Web 2.0 applications. Regarding the second category, although the research on detecting software design patterns is mature, the research on the

automated detection of web design patterns in the application model is very limited. In (Fraternali et al, 2002) authors present the Web Quality Analyzer (WQA) which automatically analyzes the conceptual schemas of Web applications and identifies the occurrences of a predefined set of WebML design patterns. By calculating metrics about their coherent use throughout the application schema, the WQA allows designers to automatically monitor the design consistency of WebML-based applications. In (Aminzadeh et al; 2010), the authors propose a method for detecting the occurrences of Human-Computer Interaction (HCI) design patterns within the design of a web application and visualize them by using UML class diagrams.

The key difference of our approach is that there is not any limitation to focus strictly on the detection of predefined design patterns (known a priori) in the conceptual model of an application. On the contrary, we provide a methodology for supporting the automated detection of all the recurrent design structures occurring within the conceptual model due to design reuse. Such structures may be a well-known design pattern or they can also be reusable design compositions (effective or not) used by the developers to accomplish certain behavior in a specific application context. These patterns can probably lead to: (i) the identification of new design patterns for handling common design problems in the CMS domain which can be used as building blocks in future designs, or (ii) they can even stand as anti-patterns in case they are evaluated to cause serious design inconsistencies. This is a promising initiative for a mechanism supporting the automated identification of design patterns and anti-patterns for the CMS domain and promoting the pattern-based CMS design and development.

3 WebML OVERVIEW

WebML (Ceri et al, 2000) is a language for modeling data-intensive Web applications by providing a set of visual primitives for defining the conceptual models that represent the organization of their contents and hypertext interfaces. The organization of the hypertext interfaces is specified by exploiting the Hypertext model which allow designers to describe the application's hypertexts for publishing and managing content as siteviews. A siteview is a specific hypertext which can be browsed by a particular group of users. Siteviews are composed of pages, which in turn include containers of elementary

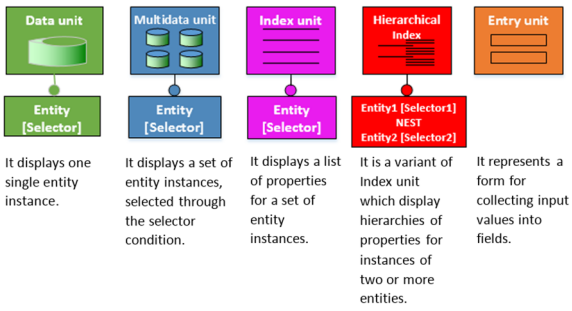


Figure 1: The WebML content units.

pieces of content called content units. The data published by a content unit are retrieved from a table of the database, which is specified by the source entity property of the unit. WebML offers a set of predefined content units such as the DataUnit, IndexUnit, MultidataUnit, ScrollerUnit, MultichoiceIndexUnit, and HierarchicalIndexUnit (some of them are presented in Figure 1) that express different ways of selecting entity instances and publishing them in a hypertext interface. Pages and units are connected with links having a twofold aim: permitting navigation and enabling the passing of parameters from the source to the destination page/unit.

By default, every WebML element has both a visual representation and an XML-based textual specification which allows specifying additional detailed properties that cannot be conveniently expressed in terms of visual notation.

4 THE METHODOLOGY

In this section, we present the methodology for automatically extracting the WebML hypertext model of a Web application (in the form of its XML specification) and its subsequent analysis with the aim of (i) identifying and evaluating by using semantic similarity techniques the occurrences of all the recurrent patterns lying within it and (ii) calculating evaluation metrics to assess if they are used consistently throughout the application model.

In order to explain the concepts of our methodology, we refer to various instances of a real web application, called the AtticaBank, which has been developed on the Joomla! platform and can be accessed at http://www.atticabank.gr/en/. This website provides information for the activities of the Attica bank which is a financial service company. In what follows, the AtticaBank website will be used to illustrate the potential of the proposed methodology.

4.1 Hypertext Model Extractor

The WebML Hypertext Model specifies the organisation of the front-end interfaces of a Web application as a set of WebML elements i.e. siteview, pages, units and links. Based on this, there are two main tasks for automatically extracting the hypertext model of a Joomla! website: (i) identify the organization of the Joomla! design elements that compose the hypertext of its HTML pages and (ii) translate them in WebML notation by producing their appropriate WebML representations.

In the context of a Joomla! website, the organization of its front-end interfaces is specified by a set of predefined structural and navigational design elements which are the components and modules. A page is composed by one component specifying the organization of its main part, and by a set of modules specifying the organization of its peripheral positions. There is a variety of components and modules categories, each one containing various types for supporting different ways of content delivery to users. In the HTML code of a page, components and modules can be found as elements. The HTML class attribute value of such a element (i.e.) specifies its style i.e. characterizes the exact type of component-module it represents. Thus, by parsing the HTML code of a page and locating the occurrences of these characteristic values within it, we can recover the page's organization as a set of Joomla! design elements. Then, the next step is to translate these design elements in WebML notation. We represent the entire website as a siteview, containing a set of WebML pages, each one corresponding to the HTML pages of the website. For the translation of the Joomla! design elements into WebML elements, we have defined appropriate WebML representations (combination of units and links) for all the types of components and modules. The complete list of the charasteristic values and WebML representations for each type of components and modules is available in (CMS Modeling, 2015). By mapping every type of components and modules found within the HTML code of a page to its corresponding WebML representation, we manage to represent the organization of the website's pages as a set of WebML elements i.e. to extract its hypertext model. The hypertext model extraction process described above is supported by a set of tools as depicted in Figure 2.

By given the URL of a website as an input, the Web Crawler crawls all the pages of the website which are then passed as an input to the Hypertext Model Extractor which parses them one-by-one in order to identify their organization as a set of components and modules. As we mentioned earlier, this is achieved by parsing the HTML code of a page and locating the occurrences of all the characteristic values for the components and modules. For example, Figure 3(a) presents the Joomla! design elements that have been identified for the "Deposits" page of the AtticaBank website. As we can see, the page consists of the "Article Category List" component which displays a list of the articles of the "Deposits" category and a set of modules such as menus, breadcrumbs etc. Once this is done for all the pages of the website, the tool translates the identified Joomla! design elements of every page into the XML textual specification of their corresponding WebML representations. Figure 3(b) presents the corresponding WebML translation of the Joomla! elements found in Figure 3(a). Finally, the tool produces an XML file as an output, containing the XML textual representations of the Joomla! elements found in all the pages of the website. This way, we can obtain the XML specification of the application's hypertext model.

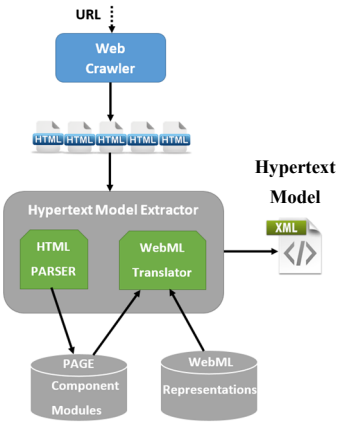


Figure 2: The hypertext model extraction process.

4.2 Identification of the Recurrent Patterns

After the extraction of the application's model, the next step is to inspect and analyze it in order to detect the potential cases of design reuse. At the hypertext level of an application, we consider as design reuse the recurrent patterns (i.e. configuration of WebML elements) within its hypertext model performing a similar functionality. Therefore, we perform a pattern-based analysis of the recovered model aiming to identify the occurrences of all the incorporated recurrent patterns and evaluate the possibility of implying design reuse. Given that the WebML

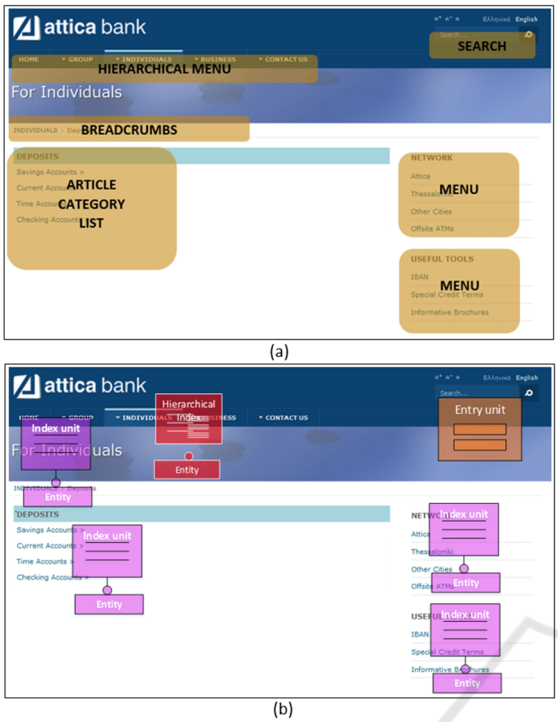


Figure 3: (a) The organization of a page in terms of Joomla! design elements. (b) The organization of a page in terms of WebML elements. HierarchicalIndexUnit is colored in red, IndexUnit is colored in fuchsia and EntryUnit is colored in orange.

elements constituting a pattern represent Joomla! components and modules, the identified patterns actually correspond to configurations of Joomla! front-end design elements which when located in a particular layout may serve a certain application purpose.

The identification of patterns (their core specifications along with their starting and termination variants) within a large collection of WebML elements, such as the application's siteview, can be reduced into the domain of graph mining and particularly to the subgraph isomorphism problem. The latter is synopsized in its general form to finding whether the isomorphic image of a subgraph exists in a larger graph, an example of which is depicted in Figure 4 (the subgraph has WebML elements on its nodes). The subgraph in Figure 4(b) is isomorphic to the subgraph in Figure 4(a). Despite the different configuration of the nodes in the two subgraphs, the edges connecting the nodes of the same colour remain the same. Table 1 contains some sequences of the nodes that are connected in the subgraphs in Figure 4.

For example, a sequence can start from node M, from which one can navigates to node I and then to node E, and so on. By observing the node sequences,

Figure 4: An example of two isomorphic WebML subgraphs. M stands for the MultidataUnit, I for the IndexUnit, D for the DataUnit and E for the EntryUnit.

one can notice that they can reveal the recurrent patterns within a graph, both their core specifications and their variants. In Table 1, there are two patterns. For example, the core specification of the first one consists of a MultidataUnit (M) which is connected to an IndexUnit (I) which in turn is connected to an EntryUnit (E). There is also a starting variant consisting of a DataUnit (D) passing content to the core specification and a termination variant consisting of a MultidataUnit (M) receiving content from the core specification. Clearly, the identification of the isomorphic subgraphs within a graph is an alternative way to obtain the identification of the incorporated recurrent patterns. Based on this, we can consider the siteview of the previously extracted hypertext model as a graph of WebML elements and attempt to identify the recurrent patterns by locating all the isomorphic subgraphs within the graph. To achieve this, we first transform the XML specification of the hypertext model into a directed graph (WebML Hypertext graph). Then, by applying a graph mining algorithm, we identify the occurrences of all the recurrent isomorphic subgraphs-patterns within this graph. The following subsections present these two tasks in detail.

4.2.1 Hypertext Model Transformation into Graph

We define the siteview of the hypertext model as a directed graph of the form G (V, E, f_V, f_E), comprising a set of nodes V, a set of edges E, a node labelling

Table 1: The sequences of the connected nodes.

Starting Variant	Pattern's Core Specification			Termination Variant
	M	I	E	
	M	I	E	M
D	M	I	E	
	I	M	D	
E	I	M	D	
	I	M	D	E

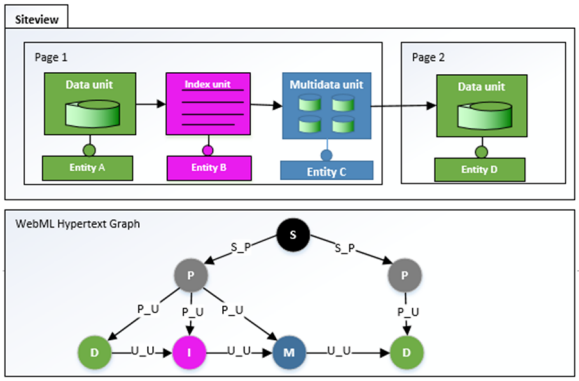


Figure 5: Transformation of a WebML hypertext composition to its graph equivalent.

function f_V: $V \rightarrow \Sigma_V$ and an edge labelling function f_E: $E \rightarrow \Sigma_E$. The function f_V assigns labels to the nodes in V from the alphabet Σ_V = {Siteview (S), Page (P), DataUnit (D), MultidataUnit (M), IndexUnit (I), HierarchicalIndexUnit (H), MultichoiceIndexUnit (MI), EntryUnit (E), ScrollerUnit (S)} which includes all the different types of WebML elements. Similarly, function f_E assign labels to the edges in E from the alphabet Σ_E = {S_P, P_U, U_P, U_U}. The label S_P denotes the containment of a page in a siteview, the label P_U denotes the containment of a content unit within a page, the label U_P denotes the link from a content unit to a page and finally the label U_U denotes the link between content units.

To transform the XML specification of the hypertext model into a directed graph, we parse the previously created XML file and create the graph as follows: we assign a node to every page and content unit of the siteview. Starting from the root node, the S node representing the siteview, we introduce edges labeled as S_P to the nodes corresponding to its WebML pages. Then, we locate edges labeled as P_U from the P (Page) nodes to the nodes corresponding to the WebML units they contain. Similarly, we locate edges with the labels U_P and U_U. An example of transforming an instance of a siteview consisting of two pages into its graph equivalent is depicted in Figure 5.

4.2.2 Mining the Recurrent Patterns

In order to identify the recurrent patterns lying within the WebML Hypertext graph, based on the approach of the subgraph isomorphic problem, we need to apply a graph mining algorithm for detecting the isomorphic subgraphs images. This problem has proven to be NP-complete. However, quite a few heuristics have been proposed to solve it, among which we have selected the most prominent one, the gSpan algorithm (Yan et al, 2002). More specifically, gSpan addresses the problem of frequent subgraph mining. Intuitively, gSpan traverses the WebML Hypertext graph G and finds all the smaller subgraphs g in G that occur frequently. A subgraph g is frequent, if its occurrence frequency in G, denoted as support(g), is no less than a minimum support threshold (minSup). In a more formal definition, the problem of frequent subgraph mining is to find any subgraph g into G so that support(g) ≥ minSup. When looking for the occurrences of a subgraph in G, the algorithm encounters except for its identical occurrences, its isomorphic images too. In this way, the graph G is analyzed in terms of its frequent subgraphs (representing recurrent patterns). To apply the gSpan on the graph, we use the Parsemis project (Philippsen, 2011) which supports an implementation of the gSpan algorithm within a graphical environment for visualizing the identified frequent subgraphs. An example can be found in Figure 6 which presents a case in which gSpan has identified a set of frequent subgraphs within the hypertext model of the AtticaBank website.

The Parsemis tool provides the identified subgraphs in a TXT file containing the configuration of the WebML elements that compose each subgraph as well as their occurrences in the graph. Then, we process this file in a way similar with the one presented in the example of Table 1 (based on the sequences of the connected nodes in the subgraphs) and identify the core specifications and the starting and ending variants of each identified pattern.

In order to examine whether the identified recurrent patterns perform similar functionality (so that we can consider them as design reuse), it is necessary to additionally inspect them by means of the content displayed by their constituent WebML units and particularly their semantic closeness. This means that if we have identified a pattern of Joomla!

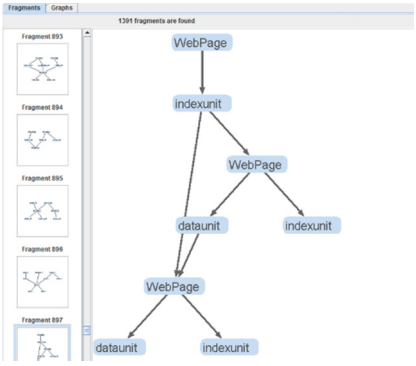
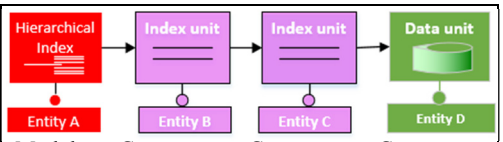


Figure 6: An instance of the Parsemis Project while detecting the frequent subgraphs in the WebML hypertext graph of the AtticaBank website.

Table 2: Average semantic similarity computation for the occurrences of an identified pattern.

PATTERN	 Module	Component	Component	Component
Occ.1	Individuals Top Menu	Individuals Deposits Categories Page	Individuals Deposits Time Accounts Category Page	Individuals Single Term Deposits Details
Occ.2	Business Top Menu	Business Deposits Categories Page	Business Deposits Time Accounts Category Page	Business Single Term Deposits Details
Sem Sim Score Occ1.-Occ.2		80%	90%	100%
AverageSemSimScore Occ1.-Occ.2				90%
Occ.3	Individuals Top Menu	Individuals Loans Categories Page	Individuals Mortgage Loans Category Page	Individuals Attica Housing Mortgage Loan Details
Sem Sim Score Occ1.-Occ.3		30%	30%	20%
AverageSemSimScore Occ1.-Occ.3				27%

front-end elements, we also have to examine the content displayed by them, for every occurrence of the pattern, in order to verify if there is any kind of common functionality performed by them. An example would be to capture a pattern used for displaying information about products of a specific product category in an e-commerce website.

The content displayed by a WebML unit is specified by its source entity property which refers to the table of the underlying database that provides content to it. In this work, we assume that we do not have access to the database of the website under study, due to the fact that this is the common scenario in real-life websites. To capture the semantic closeness of the content displayed by the pattern's WebML units (i.e. the Joomla component-module they represent) among the occurrences of the identified patterns, we have defined two metrics, the "SemSimScore" and the "AverageSemSimScore".

On the grounds that the main content of a page, published by components, is indicative of the page's semantics, the "SemSimScore" metric addresses the semantic similarity measurement of the content published by the pattern's WebML units which represent Joomla! components. This is why there are empty cells for the"SemSimScore" computations in Table 2, when it comes to measure semantic similarity among modules. Then, the "AverageSemSimScore" computes the average value of the individual "SemSimScore" values between the pattern occurences. The rationale behind this is that the content of the pages that derive from the same database's table usually has a very close semantic relation. In Table 2, we can see an example of an identified pattern performing the following behavior: from a starting point e.g. a menu link (HierarchicalIndexUnit), the user navigates to a page displaying a list of the categories of a subject such as a speficic information object type (IndexUnit), then after selecting a category, he navigates to a page displaying a list of the subcategories (IndexUnit) of the selected category and finally after selecting a subcategory, he navigates to a page displaying the details of that subcategory (DataUnit). This pattern occurs frequently in the AtticaBank website. In Table 2, we can see three occurrences of the pattern. In Occ.1, all the categories and the subcategories of the deposits for Individuals customers are displayed. In Occ.2 the same happens except for the fact that it is intended for Business customers. By comparing the semantic similarity of the content displayed by the pattern's WebML units for these two occurrences, they have an AverageSemSimScore of 90% which means that they are semantically very close. Occ.3 refers to the case of displaying all the categories and the subcategories of the loans for Individuals customers. In the same way, we can compute the AverageSemSimScore for Occ.1 and Occ.3 which is 27%, implying that these occurrences are not semantically close. This is actually expected, since their only common base is that they refer to the customer of type Individuals. The rest of the content displayed on them is irrelevant.

By using the AverageSemSimScore metric, we can obtain a safe estimation of the content semantic closeness among the occurrences of the identified patterns. In Figure 7(b), we can see the real tables of the AtticaBank database schema while in Figure 7(a), we can see the source entities of the pattern's WebML units for the occurrences Occ.1 and Occ.2. Obviously, the high semantic similarity score that we have computed for Occ.1 and Occ.2 is justified by the fact that supposing we have access to the database, the WebML units for these occurrences would have displayed content from the same database tables. We consider that these two occurrences perform similar

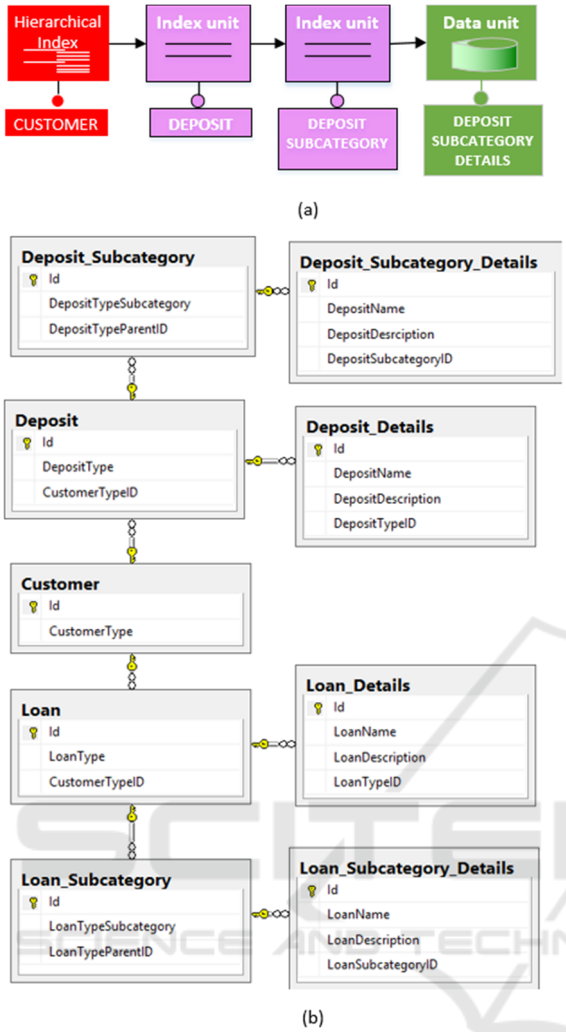


Figure 7: (a) The entities of the WebML units and (b) the AtticaBank website's database.

functionality which is to display the details of a specific type of deposits.

We compute the AverageSemSimScore metric for all the occurrences of the identified patterns (core specification and variants) and we select and store in a "Patterns Repository" only the ones having an AverageSemSimScore over 70%. The reason for this is to detect cases having a high possibility of performing a kind of similar functionality. To compute the "SemSimScore" metric, we have used the methodology proposed in (Simpson et al, 2010) for WordNet-based semantic similarity measurement. Detailed information can be found in (Simpson et al, 2010).

4.3 Evaluation of Pattern Variants Consistent Use

In this final step, we focus on evaluating the consistent design reuse. More specifically, we calculate some metrics to evaluate whether the patterns stored in the "Patterns Repository" are used consistently throughout the hypertext model. Fraternali et al. (2002) have introduced a methodology for the evaluation of the consistent application of predefined WebML design patterns within the conceptual schema of an application. We utilize this methodology and extend it in order to introduce metrics computing the consistent application of a pattern's variants throughout the hypertext model of the application. Assuming that a pattern can have N starting and M termination variants, we have defined two metrics that compute the statistical variance of the occurrences of the N starting and the M termination variants of the pattern, normalized according to the best-case variance. These metrics are called Start-Point Metric (SPM) and End-Point Metric (EPM) respectively. SPM is defined as (EPM is defined in an analogous way)

$$SPM = \sigma^2 / \sigma_{BC}^2 \quad (1)$$

σ^2 is the statistical variance of the N starting variants occurrences which is calculated according to the formula (2):

$$\sigma^2 = \frac{1}{N} \sum_{i=0}^{N} \left(p_i - \frac{1}{N} \right)^2 \quad (2)$$

where p_i is the percentage of occurrences for the i-*th* pattern variant. σ_{BC}^2 is instead the best case variance and it is calculated by the formula (2) assuming that only one variant has been coherently used throughout the application.

The last step in the metrics definition is the creation of a measurement scale to define a mapping between the numerical results obtained through the calculus method and a set of (predefined) meaningful and discrete values. According to the scale types defined in the measurement theory, the SPM metric adopts an ordinal nominal scale; each no minal value in the scale expresses a consistency level,

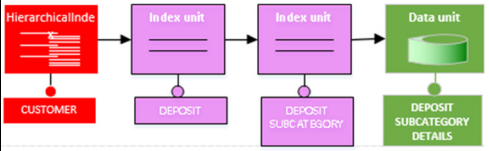
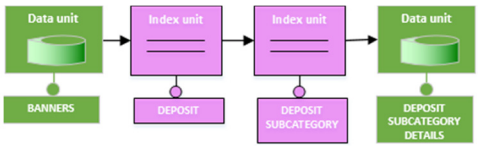
Table 3: The measurement scale for the SPM metric.

SPM range	Measurement scale value
$0 \leq SPM < 0.2$	Insufficient
$0.2 \leq SPM < 0.4$	Weak
$0.4 \leq SPM < 0.6$	Discrete
$0.6 \leq SPM < 0.8$	Good
$0.8 \leq SPM \leq 1$	Optimum

corresponding to a range of numerical values of the metrics as defined in Table 3. The same scale covers the EPM metric as well.

We compute the SPM and EPM metrics for all the occurrences of all the patterns variants and store the results in the "Results Repository". This repository contains the ranking of the pattern variants stored in the "Patterns Repository" according to their SPM and EPM values. The results are also provided in a TXT file ("Results"). An example can be found in Table 4. We consider that the core specification of the pattern consists of an IndexUnit which navigates to another IndexUnix which in turn results to a DataUnit. This is a pattern for browsing a hierarchy of categories and subcategories related to a specific information object (i.e. the Deposit object type). In Table 4, there are two starting variants of this pattern. The first one, Occ.1, consists of an HierarchicalIndexUnit passing content to the core specification, some occurrences of which we have seen in Table 2 (Occ.1, Occ.2, Occ.3). The second variant consists of a DataUnit instead of a HierarchicalIndexUnit. One occurrence of this variant can be found in Occ.4 (Table 4) in which from a starting point e.g. a banner on the Home page of the AtticaBank website, the user can have access to all the categories and the subcategories of the deposits for Individuals customers. In fact, the occurrence Occ.4 of the second variant performs the same functionality with the occurrence Occ.1 of the first variant, except for the fact that they differ on the starting point they provide to users for browsing the hierarchy of categories and subcategories. The value of the SPM metric for the first variant is about 0.68 (‘Good’) whereas the value for the second variant is 0.39 (‘Weak’). The ranking of the first variant as ‘Good’ is expected since it occurs frequently on the AtticaBank website. On the other hand, the low value of the second variant is due to the fact that it has a very limited number of occurrences. By observing the low value of the second variant on the "Results" file, developers can inspect its identified occurrences on the AtticaBank website in order to verify if it actually a design inconsistency. It is worth noting that in some cases it could happen that detected inconsistencies can be caused by conscious design choices in order to respond to specific application constraints. In the case of the AtticaBank website, despite its low SPM value, the second variant is indeed an explicit choice of the designers in order to provide a quick link to the Individuals Deposits Page for increasing its popularity. In the case of another website though, it could indicate a design mistake. Furthermore, developers can consider adding similar banners for direct navigation to other types of deposits pages, so that to enhance the application’s usability.

Table 4: Pattern variants.

VARIANT 1				
	Module	Component	Component	Component
Occ.1	Individuals Top Menu	Individuals Deposits Categories Page	Individuals Deposits Time Accounts Category Page	Individuals Single Term Deposits Details
VARIANT 2				
	Module	Component	Component	Component
Occ.4	Home Page Banner	Individuals Deposits Categories Page	Individuals Deposits Time Accounts Category Page	Individuals Single Term Deposits Details

This is just a simple example showing how the proposed methodology can work as a black box analysis of the application’s model able to highlight potential design problems to designers. We are now developing a graphical environment for visualizing the results for each pattern’s variants.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we have illustrated a model-driven approach for the automated design analysis and evaluation of CMS-based Web applications. To achieve this, we focused on evaluating the consistency of design reuse within the hypertext model of a website. We provide an automated way for extracting the hypertext model of a website which is then submitted to a pattern-based analysis for the identification of all the occurrences of the recurrent patterns lying within it. Finally, the identified patterns variants are evaluated towards their consistent use throughout the application. Most of the work

presented here can be generalized also to web applications built on other CMS platforms with slight straightforward modifications.

By applying the methodology on a website, developers can gain important information regarding its design quality. On one side, the methodology can detect effective reusable design solutions which are consistenlty used throughout an application model for solving a frequently occuring problem. Such reusable solutions can be used as building blocks for implementing certain behavior in future designs. They also facilitate the discovery of new design patterns for the CMS domain. On the other side, the methodology can also detect recurrent design constructs indicating ad-hoc forms of reuse, causing design inconsistencies and implying the need for refactoring, in order to improve the application's consistency. Developers can inspect the occurrences of such fragments on the website, as they are highlighted by the proposed methodology.

In the future, we plan to apply the methodology to a very large number of domain-specific websites for two main reasons. The first one is to better refine the methodology itself and fine-tune the currently used evaluation metrics, or even explore new ones. The second reason is to populate a central patterns repository, containing all patterns that we can possibly identify within the various websites designs. In this way, it is possible to come up with useful design guidelines for Joomla!-based websites. Our vision is to create a knowledge base of navigation and interface patterns for the CMS domain in order to form a common vocabulary among designers for solving common CMS design problems and producing quality CMS designs.

REFERENCES

Aminzadeh, N., Salim, S.S., 2010. Detecting and visualizing web design patterns, In the *Proceedings of the 2nd International Conference on Computer and Automation Engineering (ICCAE), Vol. 2, pp. 100-103.*

Aragón, G., Escalona, M. J., Lang, M., Hilera, J. R., 2013. An Analysis of Model-Driven Web Engineering Methodologies, In *International Journal of Innovative Computing, Information and Control*, Vol. 9, no. 1, pp. 413-436.

Bernstein, M., 1998. Patterns of Hypertext. In *Hypertext 98 - Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia,* Pittsburgh, PA, USA, June 20-24, 1998, pp. 21-29.

Ceri S., Fraternali P., Bongio A., 2000. Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. In the *Proceedings of WWW Conference.* Amsterdam, NL, May 2000, pp. 137-157.

CMS Modeling, (2015) Available at: http://alkistis.ceid.upatras.gr/research/modeling/CMSModelExtractor/ (Accessed: 10 November 2015).

Díaz, P., Rosson, M.B., Aedo, I., Carroll, J.M., 2009. Web design patterns: Investigating user goals and browsing strategies. In: Pipek, V., Rosson, M.B., de Ruyter, B., Wulf, V. (eds.) IS-EUD 2009. LNCS, vol. 5435, pp. 186–204. Springer, Heidelberg.

Drupal CMS (2015). Available at: https://www.drupal.org/ (Accessed: 10 November 2015).

Fraternali, P., Matera, M., Maurino A., 2002. WQA: an XSL Framework for Analyzing the Quality of Web Applications. In the *Proceedings of the 2nd International Workshop on Web-Oriented Software Technologies* – IWWOST'02. Malaga, Spain, June 10–14, 2002, pp. 46–61.

Fraternali, P., Tisi, M., 2008. Building community-based Web applications with a Model-Driven approach and design patterns. In *Murugesan, S. (ed.) Handbook of Research on Web 2.0, 3.0, and X.0: Technologies, Business, and Social Applications,* IGI Global.

Ivory, M. Y., Megraw, R., 2005. Evolution of web site design patterns. In *ACM Trans. Inf. Syst.*, 23(4):463–497.

Joomla! CMS (2015). Available at: http://www.joomla.org/ (Accessed: 10 November 2015).

Philippsen, M., 2011. *ParSeMiS - the Parallel and Sequential Mining Suite*. Available at: https://www2.cs.fau.de/EN/research/zold/ParSeMiS/index.html (Accessed: 10 November 2015).

Rigou, M., Sirmakessis, S., Tzimas G., 2006. Model Cloning: A Push to Reuse or a Disaster? In *Adaptive and Personalized Semantic Web - Proceedings of the 16th ACM Hypertext*, Springer, Studies in Computational Intelligence (SCI) Vol. 14, pp. 37-55.

Simpson, T., Dao, T., 2010. *WordNet-based semantic similarity measurement*. Available at: http://www.codeproject.com/Articles/11835/WordNet-based-semantic-similarity-measurement (Accessed: 10 November 2015).

Website patterns (2015). Available at: http://c2.com/cgi/wiki?HypermediaDesignPatternsRepository (Accessed: 10 November 2015).

Welie, M. v. (2008): Interaction Design Patterns. Available at http://www.welie.com/patterns/ (Accessed: 10 November 2015).

Yan, X., Han, J., 2002. gSpan: Graph-based substructure pattern mining. In *ICDM '02*, page 721, Washington,DC, USA, 2002. IEEE Computer Society.