Discovering Re-Usable Design Solutions in Web Conceptual Schemas: Metrics and Methodology

Yannis Panagis[1,2], Evangelos Sakkopoulos[1,2], Spiros Sirmakessis[1], Athanasios Tsakalidis[1,2], Giannis Tzimas[1,2]

[1] Research Academic Computer Technology Institute
61 Riga Feraiou str., GR-262 21 Patras, Hellas
[2] University of Patras, Computer Engineering and Informatics Department
GR-26504, Rio Patras, Hellas
{panagis, sakkopul, syrma, tsak, tzimas}@ceid.upatras.gr

Abstract. In the Internet era, the development of Web applications has impressively evolved and is characterized by a large degree of complexity. To this end, software community has proposed a variety of modeling methods and techniques. In this work, we provide a methodology and metrics for mining the conceptual schema of applications, to discover recurrent design solutions in an automatic manner. The mechanism is designed for models based on WebML, a modeling language for designing data-intensive applications. This approach, when applied in an application's conceptual schema, results in effective design solutions, as it facilitates reuse and consistency in the development and maintenance process. Furthermore, when applied to a large number of applications, it enables hypertext architects to identify templates for Web application frameworks for specific domains and to discover new design patterns extending the predefined set of patterns supported by WebML. Finally, we illustrate a validation scenario.

1 Introduction

The unprecedented adoption of Internet is setting new standards to the development of Web applications. Web applications are becoming the de facto underlying engine of any e-service, including e-learning, CSCW, e-business and e-government. Consequently, the hypertext architect has to design the application in such a way, that it can efficiently manage huge amounts of data, integrate complicated functions and sophisticated business logic. At the same time the application must provide access to users with different preferences and needs, who use a variety of access devices including mobile ones.

Novel challenges are therefore posed to developers. As the market needs increase swiftly and the use of a large number of new technologies evolves at a rapid pace, advanced Web application development becomes more and more slow, expensive and error prone, often yielding products with large numbers of defects, thereby causing serious problems of usability, reliability, performance, security and degradation of other quality of service characteristics.

Several Web application modeling methods have been proposed to tackle web development setbacks, primarily based on the key principle of separating data management, site structure and page presentation. HDM pioneered the model-driven design of hypermedia applications and influenced several subsequent proposals like HDM-lite, a Web-specific version of HDM, RMM, Strudel, and OOHDM [13].

Araneus [2] is a proposal for Web design and reverse-engineering, in which the data structure is described by means of the E-R Model and navigation is specified using the Navigation Conceptual Model (NCM). OOHDM (Object-Oriented Hypermedia Design Method) [17] is concerned with the conceptual modeling, navigation design, interface design, and implementation of hypermedia applications. Navigational contexts in OOHDM provide a rich repertoire of fixed navigation options. There also exist several proposals for using UML [4] for modeling the architecture of web applications. Some extensions to the UML notation have been proposed by Conallen [7].

In this work, WebML [5] has been utilized as design platform for the discovery methods proposed, mainly because of the robust CASE tool called WebRatio [19] that it is supported by. In fact, WebML builds on several previous proposals for hypermedia and web design language. It provides graphical, yet formal, specifications, incorporated in a complete design process, which can be assisted by visual design tools for expressing a hypertext as a set of pages made up of linked content units and operations, and for binding such content units and operations to the data they refer to. WebML is a visual language for specifying the content structure of a Web application and the organization and presentation of contents in one or more hypertexts. The first step of designing in WebML is to specify the *data schema* of the Web application, in order to express the organization of contents using E-R primitives. The next step is *Hypertext Design,* which produces schemes expressing the composition of content and the invocation of operations within pages, as well as the definition of links between pages. Apart from content publishing, WebML allows specifying data update operations, like the creation, modification and deletion of instances of an entity, or the creation and deletion of instances of a relationship. Special purpose operations, as e-mail, login, and e-payment, can also be specified.

1.1 Motivation

In a plethora of web applications the final outcome is a result of the joint design decisions made by separate groups of experts, often with diverse backgrounds. Typical examples of this practice are e-shops and e-learning environments. Furthermore, even when a modeling language such as the WebML is deployed, such is the scale of the development that a large number of coordinated developers are required to deliver the end product. These facts account for inconsistencies, debugging overheads and moderate code and design reusability.

On the other hand, the ability to detect, during the early development stages, similar design snippets or even larger design constructs that perform the same functionalities, can increase implementation consistency, application maintenance and quality. Additionally, a methodology to infer frequent constructs at the design level, when applied to the same application domain, can lead to safe conclusions as for when a specific construct constitutes a *design pattern*.

The remainder of this paper is organized as follows: Section 2 presents in detail a methodological approach for identifying reusable design solutions within the conceptual schema of Web Applications, while Section 3 illustrates a validation example of the proposed methodology in an instance of an application scenario. Finally, Section 4 provides concluding remarks and discusses future steps.

2 Methodology and Metrics for Identifying Reusable Designs

The notion of design patterns as tools that describe a piece of design experience and/or expert advice and make it reusable, was initially conceived by the architect C. Alexander, in the context of architecture and urban planning [1]: "... *Each pattern describes a problem which occurs over and over again in our environment, and then describes the core solution to that problem, in such a way that you can use this solution a million of times over...*" Nowadays, the use of patterns has been further extended in diverse domains. In the field of software engineering, design patterns are increasingly used to capture expertise in object-oriented programming [10]. More recently, design patterns have been introduced in the Web modeling field as well, for describing the navigation and structure of Web applications [3], [12], [14], [15], [16]. The availability of design patterns, which offer verified solutions to typical page configuration requirements, further facilitates the task of the hypertext architect and enforces a coherent design style over large and complicated applications, augmenting hypertext regularity and usability.

2.1 The Notion of Design Patterns within WebML

A primitive set of design patterns has already been identified in WebML, comprising compact and consistent, one-step solutions, applicable in real-life scenarios of Web applications. Patterns have been discovered for data design, by identifying typical roles of information objects within the data schema (i.e., core concepts, interconnection concepts, access facilitators), and typical data sub-schemas constructed around such roles (i.e., core, interconnection, access, personalization sub-schema) [6]. Patterns have also been defined for hypertext design, by identifying unit compositions representing typical hypertext navigation chains and *content publishing* (cascaded index, filtered index, filtered scrolled index, guided tour, indexed guided tour, object viewpoint, nested data, hierarchical index with alternative sub-pages). Moreover, WebML also introduces patterns for *content management* operations (object creation/deletion/modification, relationship creation/deletion, create/connect pattern, cascaded delete) [5].
A pattern in WebML, typically consists of a *core specification*, representing the invariant WebML unit composition that characterizes the pattern, and a number of *pattern variants*, which extend the core specification with all the valid modalities in which the pattern can start (*starting variants*) or terminate (*termination variants*). Starting variants describe which units can be used for passing the context to the core

pattern composition, while termination variants describe how the context generated by the core pattern composition is passed to successive hypertext compositions [9].

2.2 The Methodology

In this section, we present in detail a methodology for mining recurrent design solutions in the conceptual schema of applications modeled using WebML. Our objective is to capture compositions of hypertext elements (pages, units, operations, links) serving several application purposes. Examples could be the arrangement of pages, units, and links for supporting the navigation between a number of core objects, or for accessing a core object via one or more access objects and creating a new object through an operation.

These configurations are captured in the process of (or after) modeling a Web application, are complementary to the WebML predefined set of patterns and can serve as effective design solutions enabling reuse, consistency and quality improvement in the development and maintenance process. The methodology can be applied to a large number of Web applications, in order to assist in the identification of templates for Web application frameworks for specific domains and in the discovery of new design patterns extending the predefined set of patterns supported by WebML, since the process of finding new patterns involves analyzing successful applications and reverse-architecting its underlying design structure [16]. In the remainder of the paper we will refer to these configurations as “candidate patterns” or “design constructs/solutions”.

In the sequel we present in detail the seven steps of the extraction mechanism:

Steps 1 & 2: Conceptual Schema Initialization

1. Iteratively, we traverse each site view of the Web application’s conceptual schema and search for the existence of predefined WebML patterns (content publishing and content management patterns) taking into account their variants. Every pattern found is stored in a *pattern occurrences repository*, along with its starting and termination variants. The repository also stores the occurrence frequency of each pattern. Fig. 1 depicts the retrieval of a filtered index, within a site view. The above can be achieved using XSL [9]. The XSL language [20] allows writing pattern-matching rules that can be applied to an XML document for generating a new XML document. Each rule contains a matching part for selecting the target XML elements, and an action part to transform the matched elements. The XSL documents serve therefore the purpose of extracting the instances of patterns from the XML specification of the WebML conceptual schema.
2. The purpose of this step is to create a more uniform conceptual schema, thus enabling the easier extraction of design constructs in the steps to follow. Taking into account the various predefined WebML pattern variants, and utilizing XSL rules, we substitute, where possible, the variants found within each site view with a default pattern variant. The default pattern variant is the one having the maximum occurrence frequency (e.g. if we find a modify pattern and its termination variant having the larger occurrence frequency is the same page termination variant, we

use it as the default pattern and substitute all the other variants).[1] In case that more than one pattern is assigned the maximum frequency, we choose the first found.

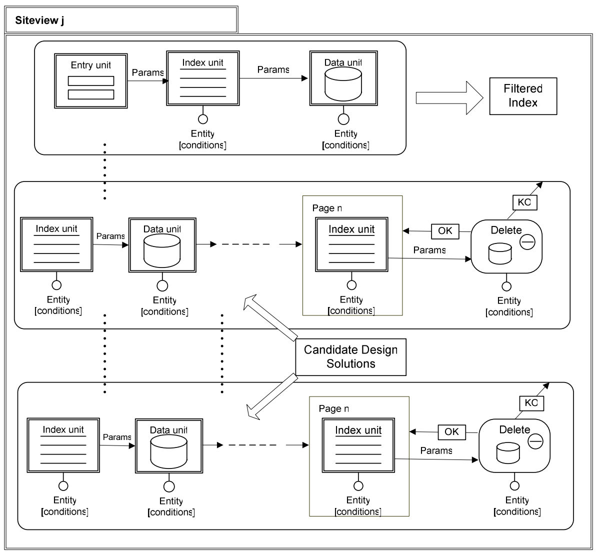

Fig. 1. Retrieval of a predefined WebML pattern and a design construct within a site view

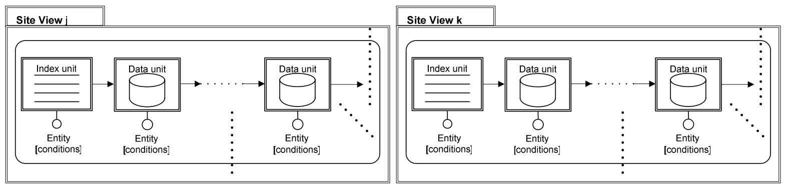

Fig. 2. Retrieval of a design construct from different site views

Step 3: Design Solutions Extraction

3. We traverse each area, sub-area and page of all the newly generated site views, in order to locate identical configurations of hypertext elements along with their variants, either within a site view or among different site views, using the methodology presented in section 2.3. The configurations retrieved should not already belong to the predefined WebML set of patterns but may contain one or more of them. The notion of a variant in this case follows the definition of the WebML pattern variants presented in section 2.1. This way, we extract a first set of design constructs. Fig. 1 depicts the retrieval of a design construct within a site view, while Fig. 2 represents the identification of such a design construct retrieved from two distinct site views. The constructs identified are stored in a repository (of candidate patterns), along with their frequency and a list of other parameters described in section 2.5.

[1] This step requires the designer's intervention to assure that the substitution does not lead to inconsistencies in the conceptual schema.

Steps 4, 5 & 6: Extending the sets of design solutions & their variants

4. Following the procedure described in step 2 and taking into account the design constructs and variants' definitions stored in the repository, we compute a new XML definition of each site view, by substituting -where possible- the variants with the default construct variant, aiming to create a more canonical schema. We then repeat step 3 in order mine a larger set of hypertext configurations that have not already retrieved in the previous step.
5. Based on the procedure introduced in section 2.4 we try to capture larger –possibly combinations of- design constructs in order to enrich the repository with the maximum number of design solutions.
6. We examine every pattern within the repository and in case it contains a predefined WebML content management pattern, but one or more of the remaining patterns is missing, we add the respective missing ones. For instance, if we locate a design construct containing a create pattern, we complement it by adding the modify and/or delete pattern(s).

Step 7: Design solutions evaluation & ranking

7. It is obvious that the number of design solutions obtained by the above methodology can be very large if applied to a complex Web application or even worse to a number of applications. Thus a first evaluation has to be performed in order to decrease their number, and provide a first level of ranking. An analytical method accomplishing that is presented in section 2.5.

Upon the completion of the above steps, the hypertext architect has access to a repository that contains a set of design solutions along with their variants. This library is composed of basic design configurations and combinations of larger ones that can be extended (in terms of usage) by defining new variants.

3.3 Automated Extraction of Design Solutions

In this section we describe a methodology for construct (compositions of hypertext elements) mining in the site views of a WebML fabricated web application. This approach is heavily based on graph mining algorithms. Intuitively, after modeling the site views as directed graphs the task is to detect frequently occurring induced subgraphs. The problem in its general form boils down to finding whether the isomorphic image of a subgraph in a larger graph exists. The latter problem is proved to be NP-complete [11]. However, graph mining appears in many contexts including bioinformatics and chemistry and therefore quite a few heuristics have been proposed to face this problem. The most prominent approaches include *gSpan* [22], *CloseGraph* [21] and *ADI* [18]. In the following we provide some notation prior to reducing the problem to graph mining.

We define a site view as a directed graph, $G(V, E, f_V, f_E)$, comprising of a set of nodes V, a set of edges E, a node-labeling function f_V: $V \rightarrow \Sigma_V$, and an edge-labeling function f_E: $E \rightarrow \Sigma_E$. f_V assigns letters drawn from an alphabet Σ_V to the site view nodes, whereas f_E has the same role for edges and the edge alphabet Σ_E. Σ_V has a different letter for each different WebML element, where "element" includes content units, operations, pages, areas, etc. Correspondingly Σ_E comprises of all the different kinds of links. We demand that units in WebML do not exactly correspond to nodes in *V* and the same is true for links between units and edges in *E*.

This choice was dictated by the rather complicated WebML conceptual model. We have to model the fact that a hyperlink can e.g. point to a data unit as well as to a hypertext *containing* several data units. Furthermore, links can also be classified into contextual and non-contextual, not to mention that a design construct can generally span different hypertexts. Therefore, before applying any graph mining technique, we preprocess the site view into its graph representation as follows: We process the WebML application definition and assign each unit and operation a letter according to its type. We install edges between units and label each edge with a 'C' (contextual), or with a 'N' (non-contextual). As a second step we map each page, area, etc. into a separate node. An edge is introduced between e.g. a page-node and the nodes corresponding to elements it contains. This containment edge is labeled with a special letter 'c', to denote containment. Note that arbitrary containment sequences can exist. A transformation example is depicted in Fig. 3.

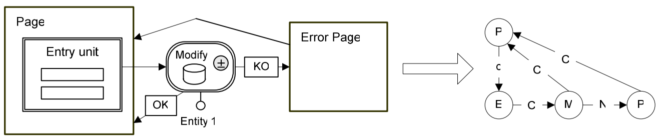

Fig. 3. Transformation of a WebML pattern to its graph equivalent

Once having transformed all site views with the same methodology, design construct identification is reduced to mining frequent subgraphs of the site view database. The latter can be accomplished with any of the methodologies in [18], [21], [22] provided the desired *support*[2] is given.

3.4 A Mechanism to Acquire Larger Design Solutions

A more complex case is the detection of implicitly interconnecting design solutions. It is about the cases of the proposed methodology step 4. The main concept is that new broader design solutions appear when taking into account the intermediate constructs as parts of the already detected variants. A minimal approach would be to look for couples of constructs that are interconnected through a single link or unit. However, we broaden the discovery procedure to include any number of constructs within a site view interconnected through intermediate structures. We think of larger design solutions to have identified constructs as a dominating part of them. As a result, we query for interconnection variants that are smaller than any other configuration involved in the larger design solution. Therefore, we utilize these constructs that include, at most, the minimum units among the configurations involved. In this way, we avoid exhaustive iterations or racing conditions. In worst case, a whole site view would be evaluated as candidate design solution, but it will be rejected following the metrics given in the next section.

[2] The minimum percentage of occurrences in the entire database.

3.5 Design Solutions Evaluation Metrics

To provide metrics in the design solutions' identification process, several evaluation factors are taken into consideration such as:

- *appearance f* denotes design solutions' frequency which is comprised by: a) overall number of appearing instances, f_o, b) appearance in f_α areas, c) appearance in f_σ site views, d) appearance in f_π pages.
- *population p* denotes number of WebML elements involved defined as: a) number of content units, p_δ, b) number of links, p_ι, c) number of operation units (including generic operations defined by the designer), p_{op}.
- *fragmentation* φ represents the number of pages hosting the design solution.
- *entities involvement e* represents number of data model entities participating in the design solution considered per conceptual factor as: a) entities belonging to the core sub-schema, e_c, b) entities belonging to the personalization sub-schema, e_{per}, c) entities belonging to the access sub-schema, e_{acc}. [3]

We define the notions of *importance*, *complexity* and *semantic value* as follows:
The *importance* V_j of the design solution *j* is:

$$V_j = f_j \, x \, p_j \, . \tag{1}$$

$$\text{where: } f_j = \frac{f_{o,j}}{f_{o,\max}} + \frac{f_{a,j}}{N_\alpha} + \frac{f_{\sigma,j}}{N_\sigma} + \frac{f_{\pi,j}}{N_\pi}, f_j \in [0,4] \, . \tag{2}$$

The notation f_{max} is the maximum value of the corresponding metric and N is the corresponding sum of areas, site views and pages overall in a specific WebML definition.

$$p_j = \frac{p_{\delta,j}}{p_{\delta,\max}} + \frac{p_{i,j}}{p_{i,\max}} + \frac{p_{op,j}}{p_{op,\max}}, p_j \in [0,3] \, . \tag{3}$$

The *complexity* d_j of the design solution *j* is:

$$d_j = \frac{1}{\phi_j} \, . \tag{4}$$

The semantic value e_j of the design solution *j* is:

$$e_j = \frac{e_{c,j}}{e_{c,\max}} + \frac{e_{per,j}}{e_{per,\max}} + \frac{e_{acc,j}}{e_{acc,\max}}, e_j \in [0,3] \, . \tag{5}$$

Overall the impact of a design solution j in a WebML conceptual schema is given by:

$$I_j = \frac{e_j}{\phi_j} V_j \, . \tag{6}$$

After computing the factor I_j for every design solution *j* in a specific conceptual schema, results are presented in a descending order to depict the most important and effective design solution on the top. The same procedure can be applied repeatedly to site views from different applications in order to detect similar "pattern" implementation behaviors and group the results.

[3] We don't include the interconnection sub-schema because it refers to entities relations.

4 Exemplifying the Methodology

Due to space limitations we will exemplify a fragment of the methodology, referring to an instance of a multinational enterprise-intranet, in which we identify a candidate design solution. The data model of the application is rather simple and is omitted. We exemplify the third and sixth step of the methodology.

In the example depicted in figures 4 and 5, we capture a candidate design solution, by traversing two distinct site views of the intranet application. The first is a fragment of an employee's site view (Fig. 4), depicting a *News* area and the second is a fragment of the manager's site view including a *Forums* area (Fig. 5).

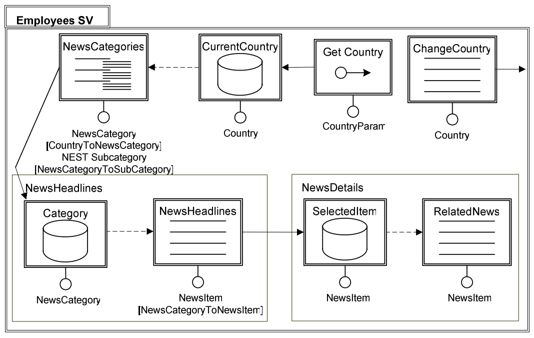

Fig. 4. A fragment of the employees site view depicting the News section[4]

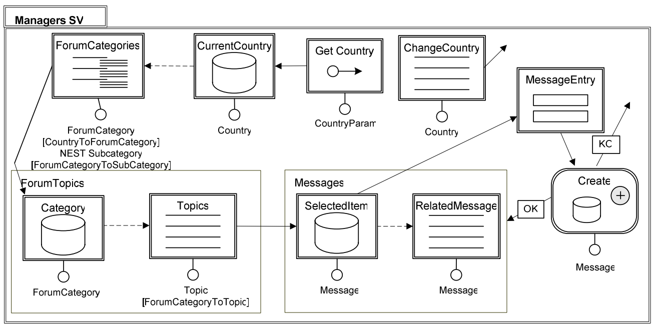

Fig. 5. A fragment of the managers site view depicting a Forum section

Comparing the two site views, we can easily identify the existence of a candidate design solution. It consists of the entire composition of WebML elements contained in the employees site view. When applying the methodology, this design construct is stored in the repository. Moreover, in the case of the manager's site view, one variant of the previously identified construct can be extracted. This variant extends the design solution with an object creation pattern (as shown in Fig. 5, the "*SelectedItem*" data unit is linked with an entry unit used to supply values to a create unit).

[4] *Get Country* is a *get unit* enabling the retrieval of a global parameter handing the applications multinational support.

Once variants containing content management patterns have been retrieved, we should extend the repository with variants derived by the missing content management patterns. For instance, a variant containing the object deletion pattern in place of the object creation pattern should be stored in the repository, as well as all the possible combinations computed using all the content management patterns.

Thus, although we have retrieved only one common navigation chain, we have constructed and stored in the repository more variants. Moreover, we have possibly identified a candidate design solution for the public information exchange within the enterprise.

5 Conclusions and Future Work

This paper illustrated a methodology and provided metrics for discovering recurrent design solutions within an applications conceptual schema modeled using WebML. This methodology when applied to a large number of WebML application schemas can form the basis for the identification of templates in specific domain Web application frameworks and become a valuable tool for hypertext architects for the identification of Web design patterns.

The design solutions extracted can complement WebML predefined patterns in the process of modeling or redesigning an application providing effectiveness, reusability and consistency. Moreover, our methodology extends the quality evaluation framework presented in [8], [9] by means of providing a mechanism for capturing project data about the recurrent use of some design solutions, through the automatic analysis of XML application schemas.

In the future, we plan to extend the methodology by providing more precise metrics for the design solutions extraction taking into account a larger number of parameters quantifying the semantic context impact on the design configurations identified. Moreover, we plan to apply the methodology on a large number of Web applications, in order to refine the methodology and fine-tune the design solutions evaluation metrics.

References

1. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A Pattern Language. Oxford University Press, New York (1997)
2. Atzeni, P., Mecca, G., Merialdo, P.: Design and Maintenance of Data-Intensive Web Sites. Proc. EDBT. (1998) 436-450
3. Bernstein, M.: Patterns of Hypertext, Proc. of HyperText'98. Pittsburgh PA (1998)
4. Booch, G., Jacobson, I., Rumbaugh, J.: The Unified Modeling Language User Guide. The Addison-Wesley Object Technology Series (1998)
5. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kauffmann (2002)
6. Ceri, S., Fraternali, P., Matera, M.: Conceptual Modeling of Data-Intensive Web Applications. IEEE Internet Computing, 6(4), (2004) 20-30
7. Conallen, J.: Building Web Applications with UML. Addison-Wesley, Reading MA (1999)

8. Fraternali, P., Matera, M., Maurino, A.: Conceptual-Level Log Analysis for the Evaluation of Web Application Quality. Proc. of IEEE LA-Web Conference. Cile (2004)
9. Fraternali, P., Matera, M., Maurino, A.: WQA: an XSL Framework for Analyzing the Quality of Web Applications. Proc. of IWWOST'02. Malaga Spain (2002)
10. Gamma, E., Helm, R., Johnson, R., Vlissedes, J.: Design Patterns - Elements of Reusable Object Oriented Software. Addison Wesley (1995)
11. Garey, M.,R., Johnson, D., S.: Computers and Intractability: A guide to NP-Completeness. Freeman, New York (1979)
12. Garzotto, F., Paolini, P., Bolchini, D., Valenti, S.: Modeling-by-Patterns of Web Applications. In Proceeding of the ER'99 Workshop "World Wide Web and Conceptual Modeling". Paris France (1999) 293-306
13. Montero, S., Diaz, P., Aedo, I.: Requirements for Hypermedia Development Methods: A Survey of Outstanding Methods. Proc. of the 14th International Conference on Advanced Information Systems Engineering. (2002) 747 – 751
14. Nanard, M., Nanard, J., Kahn, P.: Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. In Proc. of ACM Hypertext'98. Pittsburgh, PA (1998) 11-20
15. Schwabe, D, Esmeraldo, L., Rossi, G., Lyardet, F.: Engineering Web Applications for Reuse. IEEE Multimedia. Vol. 8, Issue1. (2001) 20-31
16. Schwabe, D., Garrido, A., Rossi, G.: Design Reuse in Hypermedia Applications Development. In Proc. of ACM Hypertext '97. Southampton, UK (1997) 57-66
17. Schwabe, D., Rossi, G.: An Object-Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems (TAPOS). vol. 4. no. 4, (1998) 207-225
18. Wang, C., Wang, W., Pei, J., Zhu, Y., Shi, B.: Scalable Mining of Large Disk-based Graph Databases. In Proc. ACM KDD04. (2004) 316-325
19. WebRatio: http://www.webratio.com
20. XSL: Extensible Style sheet Language. W3C Recom. http://w3.org/TR/XSL/. (2001)
21. Yan, X., Han, J.: CloseGraph: mining closed frequent graph patterns. In Proc. of KDD03. (2003) 286-295
22. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In In Proc. of Int. Conf. on Data Mining (ICDM'02). Maebashi (2002) 721-724