

internal quality is a key issue for an application's success, it is important that it is dealt with through a design view, rather than an implementation view only.

II. MODEL CLONING & MODEL SMELLS

Restructuring, refactoring and code cloning are well known notions in the software community. According to the reverse engineering taxonomy of Chikofsky and Cross [14], *restructuring* is defined as: "... the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics). A restructuring transformation is often one of appearance, such as altering code to improve its structure in the traditional sense of structured design. While restructuring creates new versions that implement or propose change to the subject system, it does not normally involve modifications because of new requirements. However, it may lead to better observations of the subject system that suggest changes that would improve aspects of the system."

In the case of object-oriented software development the definition of *refactoring* is basically the same: "... the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure" [15]. The key idea here is to redistribute classes, variables and methods in order to facilitate future adaptations and extensions.

Code cloning or the act of copying code fragments and making minor, non-functional alterations, is a well known problem for evolving software systems leading to duplicated code fragments or *code clones*. Code cloning can be traced by *code smells* that is, certain structure in code that suggests the possibility of refactoring [15].

Roundtrip engineering has reached a level of maturity that software models and program code can be perceived as two different representations of the same artifact. With such an environment in mind, the concept of refactoring can be generalized to improving the structure of software instead of just its code representation.

There is a variety of techniques for supporting the process of detecting code cloning in software systems. Some of them are based on string and token matching ([16], [17], [18], [19], [20], [21], [22]), some others on comparing sub-trees and sub-graphs ([23], [24], [25]), while others are based on metrics characterization ([26], [27], [23], [28], [29]). Moreover, there are a large number of tools that mine clones in source code and support a variety of programming languages such as C, C++, COBOL, Smalltalk, Java, and Python.

Clone mining in Web applications was first proposed by Di Lucca et al. [30], who study the detection of similar HTML pages by calculating the distance between page objects and their degree of similarity. Static page clones can also be detected with the techniques proposed by Boldyreff and Kewish [31] and Ricca and Tonella [32], with the purpose of transforming them to dynamic pages that retrieve their data

form a database. Despite that, the decreasing percentage of Web sites that merely publish static content and the current shift towards Web applications with high degree of complexity, lead to the need to introduce new techniques, capable of coping with the problem. In the specific domain of Web application modeling, the notion of cloning has not yet been introduced. Even though a few model-level restructuring techniques have been proposed, this certain issue remains open for the scientific community [3]. Moreover, the existing techniques are based exclusively on UML as the modeling language and there is no technique based on one of the rest of Web application modeling languages and methods.

In the past, a number of research attempts have been conducted in the field of refactoring applications based on their design model. Most of them focus on standalone software artifacts and deploy UML to perform refactoring [2]. But despite the popularity of model-driven methodologies, there is an absence of assessment/analysis throughout the design and development process.

In a previous work [33] we extended the notion of code cloning to the modeling level of a Web application. Analogously to code cloning, we introduced the notion of *model cloning* as the process of duplicating, and eventually modifying, a block of the existing application's model that implements certain functionality. This ad-hoc form of reuse occurs frequently during the design process of a Web application. Moreover, *model smells* are defined as certain blocks in the Web application's model implying the possibility of refactoring.

In this paper we provide a methodology and a tool supporting the evaluation of the conceptual schema of an application, by means of the design features incorporated in the application model. The objective is to capture cases (i.e. model clones) which have different design, but produce the same functionality, thus resulting in inconsistencies and ineffective design and may have been caused by inappropriate forms of model reuse.

The evaluation of the conceptual schema is performed in two steps of inspection: a first level evaluation of the hypertext compositions used in the hypertext design, and a second level evaluation of data manipulation and presentation to the user.

The proposed methodology can be deployed either in the process of designing an application or in the process of re-engineering it. In this work, WebML has been utilized as the design platform for the methods and tool proposed, mainly due to the fact that it supports a concrete framework for the formal definition of data intensive Web Applications and the fact that it is supported by WebRatio [34], a robust CASE tool.

The remaining of this paper is structured as follows: Section III provides a brief overview of WebML and its basic notation, section IV introduces the proposed methodology for mining model clones in the conceptual schema of an application, whereas section V illustrates the design and functionality of the implemented tool that applies the methodology. Finally, section VI concludes the paper and discusses future steps.

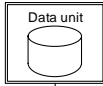
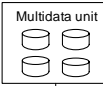
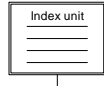
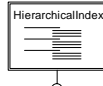
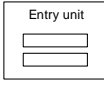
III. WEBML: A BRIEF OVERVIEW

WebML is a conceptual model for Web application design [35]. It offers a set of visual primitives for defining conceptual schemas that represent the organization of the application contents and the hypertext interface. These primitives are also provided with an XML-based textual representation, which allows specifying additional detailed properties that cannot be conveniently expressed in terms of visual notation. The organization of data is specified in WebML by exploiting the E-R model, which consists of *entities* (defined as containers of data elements) and *relationships* (defined as semantic connections between entities). WebML also allows designers to describe hypertexts for publishing and managing content, called *site views*. A site view is a specific hypertext, which can be browsed by a particular class of users. Within the scope of the same application, multiple site views can be defined.

Site views are internally organized into hypertext modules, called *areas*. Both site views and areas are composed of *pages*, which in turn include containers of elementary pieces of content, called *content units*. Typically, the data published by a content unit are retrieved from the database, whose schema is expressed by the E-R model. The binding between content units (and hypertext) and the data schema is represented by the *source entity* and the *selector* defined for each unit. The source entity specifies the type of objects published by the content unit, by referencing an entity of the E-R schema. The selector is a filter condition over the instances of the source entity, which determines the actual objects published by the unit. WebML offers predefined units, such as data, index, multidata, scroller, multichoice index, and hierarchical index (some of them are presented in Table 1), that express different ways of selecting entity instances and publishing them in a hypertext interface.

To compute its content, a unit may require the “cooperation” of other units, and the interaction of the user. Making two units interact requires connecting them with a *link*, represented as an oriented arc between a source and a destination unit. The aim of a link is twofold: permitting navigation (possibly displaying a new page, if the destination unit is placed in a different page), and enabling the passing of parameters from the source to the destination unit.

Table 1. Some basic WebML elements. The complete set is listed in [35].

Data unit	Multidata unit	Index unit	HierarchicalIndex	Entry unit
				
Entity [Selector]	Entity [Selector]	Entity [Selector]	Entity1 [Selector1] NEST Entity2 [Selector2]	
Displays a set of attributes for a single entity instance.	Displays a set of instances for a given entity.	Displays a list of properties of a given set of instances.	Displays index entries organized in a multi-level tree.	Displays forms for collecting input data into fields

Finally, WebML models the execution of arbitrary business actions, by means of *operation units*. An operation unit can be

linked to other operation or content units. WebML incorporates some predefined operations (enabling content management) for creating, modifying and deleting the instances of entities and relationships, and allows developers to extend this set.

IV. THE METHODOLOGY

In what follows we present a quick overview of the methodological approach for mining potential model clones at the conceptual schema of a Web application. A more detailed description can be found in [33]. The methodology comprises three distinct phases.

In the first phase, we transform the Web application’s conceptual schema into a set of directed graphs, representing the navigation structure and the distribution of content among the areas and pages of the application. This forms the basis for the information extraction mechanism required for the next phase. Then, we extract potential model clones and information related to the navigation and semantics of the application by utilizing graph mining techniques, and finally, in the third phase we provide a first level categorization of the potential model clones according to a number of criteria.

A. Conceptual Schema Transformation

In this phase the application’s conceptual schema is preprocessed in order to provide the means for the extraction of potential model clones. Assuming an application comprising a number of site views, we construct a first set of graphs representing the navigation, the content presentation and the manipulation mechanisms of the application. More specifically, we define a site view 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-labeling function $f_V: V \rightarrow \Sigma_V$, and an edge-labeling function $f_E: E \rightarrow \Sigma_E$. Function f_V assigns letters drawn from an alphabet Σ_V to the site view nodes, whereas f_E operates likewise for links and the edge alphabet Σ_E . Σ_V has a different letter for each different WebML element (content units, operations, pages, areas, etc).

Correspondingly, Σ_E consists of all the different kinds of links (contextual, non contextual, transport & automatic). Besides the predefined WebML links, we introduce a special kind of edge (labeled ‘c’) in order to represent the containment of content units or sub-pages in pages, as well as pages, sub-areas and operation units in areas. Note that there can be arbitrary containment sequences. A transformation example is depicted in Fig. 1 (Transformation A), where we transform a page containing several content units, interconnected by a number of contextual links.

Following a similar procedure, for every site view of the hypertext schema we create a second graph representing the data distribution within each area, sub-area and page, thus constructing a second set of graphs. In this case we define a site view as a directed graph of the form $Q(N, L, f_N, f_L)$, comprising a set of nodes N , a set of edges L , a node-labeling function $f_N: N \rightarrow \Sigma_N$, and an edge-labeling function $f_L: L \rightarrow \Sigma_L$.

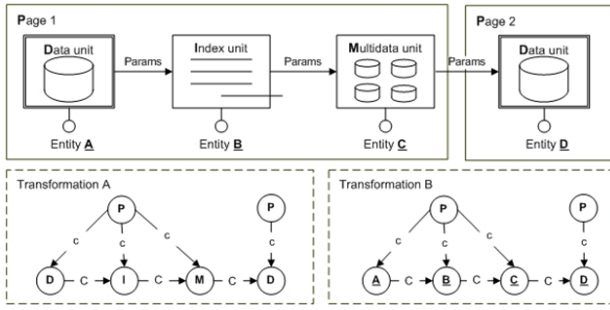


Figure 1. Transformation of a WebML hypertext composition to its graph equivalents

Function f_N assigns letters drawn from an alphabet Σ_N to the site view nodes, whereas f_L has the same role for links and the edge alphabet Σ_L . Σ_N has a different letter for each different source entity used by the WebML elements comprising the hypertext schema, as well as for the pages, sub-pages, areas and sub-areas of the site view. Σ_L comprises all the different kinds of WebML links in order to model the context navigation within the hypertext schema. As in the previous transformation, we also introduce edges denoting containment. A transformation example is depicted in Fig. 1 (Transformation B).

B. Potential Model Clones Extraction

Having modeled the navigation, content presentation and manipulation mechanisms of the application, as well as the data distribution within each site view, the next step is to capture model smells.

We traverse the first set of graphs constructed in the previous phase, in order to locate identical configurations of hypertext elements (subgraphs), either within a graph representing a single site view or among graphs representing different site views. The recovery of the various configurations can be achieved using 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 is synopsised to finding whether the isomorphic image of a subgraph exists in a larger graph. The latter problem has proved to be NP-complete (Garey and Johnson 1979) [36]. However, quite a few heuristics have been proposed to face this problem with the most prominent such approaches being the *gSpan* [37], the *CloseGraph* [38] and the *ADI* [39]. Any of the above approaches can be utilized for extracting the hypertext configurations.

Likewise, employing the same graph mining techniques, we traverse the second set of graphs in order to locate identical configurations of data elements (source entities) along with their variants.

Finally, we also locate compositions of identical hypertext elements referring to exactly the same content but interconnected with different link topologies. Ignoring the edges in the first set of graphs (except from those representing containment) we mine identical hypertext configurations within a graph or among graphs. Then, we filter the sets of

subgraphs acquired utilizing the information represented in the second set of graphs (source entities), and keep those compositions that refer to the exact same data.

C. Potential Model Clones Categorization

In this phase, we categorize all the retrieved subgraph instances, in order to facilitate the quality evaluation procedure of the overall application conceptual schema.

More precisely, for every instance of the hypertext configurations mined in the first case of graphs, we make a first level categorization according to the source entities and attributes that the WebML elements of the configurations refer to. To accomplish that, we utilize the information provided by the XML definition of each site view, where there is a detailed description of the source entities and the selectors of each element included in the site view [12].

For a specific configuration retrieved, we categorize its instances in the various site views of the application as configurations constituted by WebML elements referring to:

- exactly the same source entities and attributes,
- exactly the same source entities but different attributes (in the worst case, the only common attribute is the object identifier or OID),
- partially identical source entities (i.e. Fig. 2),
- different source entities.

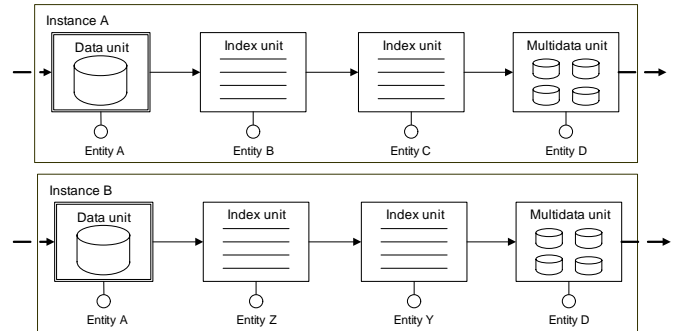


Figure 2. Categorizing potential model clones at the Hypertext Level.

We also categorize (exploiting the XML definitions) every instance of the data element configurations acquired by the graphs representing the data distribution as configurations constituted by source entities utilized by:

- different WebML elements,
- similar WebML elements, that is elements of the same type such as composition or content management (e.g. in Fig. 3, entity A is utilized by two composition units, a multidata and an index).
- identical WebML elements.

The last category captures exactly the same hypertext configurations as the first case of the previous categorization.

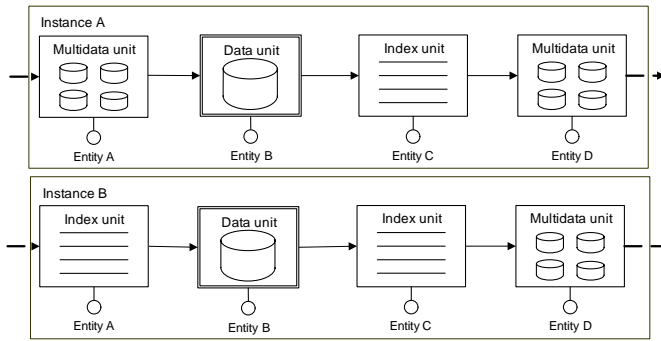


Figure 3. Categorizing potential model clones at the Data Level.

Finally, potential model clones are also the sets of hypertext configurations retrieved in the third step of the previous phase, where compositions of identical WebML elements referring to common data sources, utilizing different link topologies, have been identified (Fig. 4).

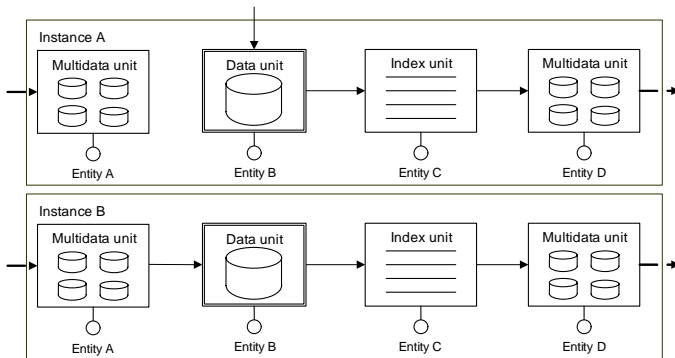


Figure 4. Potential model clones based on link topology.

Having identified the potential model clones at the hypertext and data representation level, we can provide metrics for categorizing the application conceptual schema through a quality evaluation which specifies factors (referring to the identified clones) denoting possible need for refactoring. The evaluation is threefold: First, we evaluate the model clones by means of consistency, based on their variants throughout the conceptual schema. Second, we quantify the categorization based on the similarity among potential model clones and third we make an evaluation based on the topology of links. A detailed description of such metrics, as well as refactoring proposals can be found in [33].

Due to space limitations in what follows we illustrate the tool's design and functionality covering a part of the first two phases of the methodology.

V. MODEL CLONE EXTRACTION TOOL

A. Implementation Details

The implemented tool takes as input the XML definition of one or more site views of a Web application's design and transforms it into one or more graphs.

The first transformation executes as follows: each WebML

element (i.e. index unit, data unit, etc.) is represented by a graph node and the connection between two units by a graph edge. Next, all occurrences of repeated subgraphs in the initial graph(s) are identified and highlighted. Finally, the tool outputs statistics showing the number of subgraph occurrences, along with the corresponding graph sizes.

The second transformation is similar with the first with the difference that each graph node represents a source entity. The tool highlights all occurrences of subgraphs with the exact same source entities and outputs the respective statistics.

Finally, in the third transformation each graph node represents the type of each WebML element (i.e. index unit, data unit, etc) and the corresponding source entity in order to locate compositions of identical hypertext elements referring to the exact same content.

The interface of the tool allows a stepwise execution of the successive components described below:

- **Graph generation.** Presents a dialog box asking to locate the XML definition file to be used as input to *gSpan* [37] for finding all subgraphs of the generated graph. Moreover, the file to be fed to *graphGrep* [40] (a program that locates subgraph occurrences in a graph) is also generated during this phase, along with a Visio representation of the initial graph (in form of a flowchart and using WebML elements).
- **Subgraph Statistics.** A set of statistics concerning the identified subgraphs is generated.
- **Subgraph identification.** All identified subgraphs are highlighted (in turn) in the Visio representation of the initial graph.

The tool was implemented using the Microsoft Visual Studio C# .NET environment, Microsoft Office Visio 2003, the interop library for the communication between Visual Studio and Visio, as well as *gSpan* and *graphGrep*, and its architecture is depicted in Fig. 5. The above implementation configuration was imposed by the fact that the source code of *WebRatio* [34] was not available. Thus, implementation was based on the XML definition of the conceptual schema, and the visual representation deployed Microsoft Visio.

gSpan takes as input a file, which includes all the nodes and the edges of a graph. It is executed in a Linux environment and outputs a file including all subgraphs of the initial graph.

graphGrep is compiled in Windows via the Cygwin platform and it is called by the application using as input two files; the first file contains one or more graphs representing the conceptual schema and the second file a subgraph to be located (query subgraph). The output of this program is a file that shows in which graph(s) and at which place(s) the subgraph was found.

Edge direction is a crucial requirement for our analysis, but neither *gSpan*, nor *graphGrep* support directed graphs. In order to overcome this limitation an auxiliary node is inserted between each pair of nodes named after the initials of the types of the two nodes.

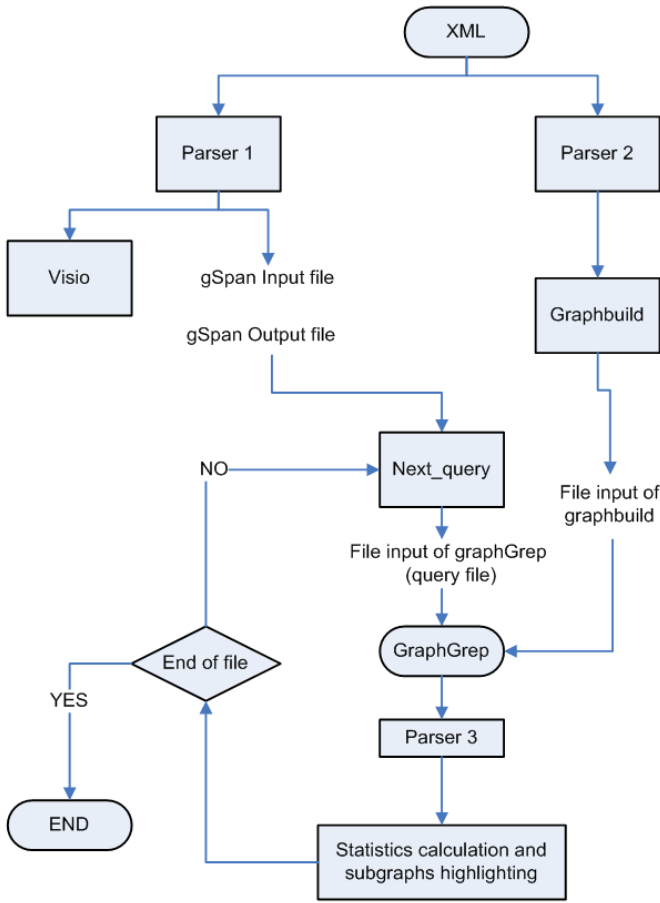


Figure 5. Architecture of the Model Clone Extraction Tool.

For instance, to model an edge directed from node d to node i (Fig. 6a), a new node named di is inserted between the two nodes (Fig. 6b). These auxiliary nodes are filtered out from graph representations that get generated by the tool, as well as from the produced statistics.

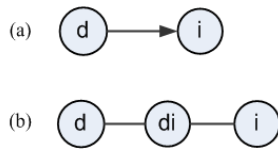


Figure 6. Modeling directed graphs in gSpan and graphGrep.

B. An Exemplifying Paradigm

Given the XML definition file of an instance of a site view, as shown in Fig. 7, the tool generates a graph representation in Visio file format as depicted in Fig. 8, where P stands for 'page', I for 'index unit', D for 'data unit', M for 'multidata unit', c for 'contains', n for 'non contextual' and C for 'contextual'. Moreover, the tool creates the files to be used as input to gSpan and graphGrep respectively.

Fig. 9 presents the generated WebML representation of the initial .xml file. gSpan outputs a file containing the subgraphs of the initial graph. This file is fed to the second component of the tool.

In Table 2 we see the total number of subgraphs identified in

the initial XML definition file of the overall web application, sorted by their frequency of appearance, along with their size. Fig. 10 and 11 present the second and the third representation of the initial .xml file.

In Fig. 11 the label of each node consists of a letter declaring its type (d for data unit, I for index unit, etc.) and a unique number identifying the entity.

In Tables 3 and 4 we see the statistics showing the number of subgraph occurrences along with the graph sizes in the second and third case respectively. Fig. 12 depicts subgraph $I-D$ as highlighted (in the second case) in the initial graph. Concluding, in Fig. 13 subgraph $P-Artist$ is highlighted (in the third case) in the initial graph in Visio.

In order to assist the designer in locating model clones, the identified subgraphs get highlighted in the visual representation of the initial graph in turn (each time the user clicks, a different subgraph is highlighted). Moreover, each subgraph is presented with a different color in order to be more easily discernable.

```

<?xml version="1.0" standalone="yes" ?>
<root>
<INDEXUNIT id="AllArtists" entity="Artist">
  <HYPERLINK id="link1" to="ArtistData"/>
</INDEXUNIT>
<DATAUNIT id="ArtistData" entity="Artist">
  <HYPERLINK id="link2" to="ArtistShortData"/>
</DATAUNIT>
<DATAUNIT id="ArtistShortData" entity="Artist">
  <INFOLINK id="link3" to="AlbumIndex"/>
</DATAUNIT>
<INDEXUNIT id="AlbumIndex" entity="ArtistToAlbum">
  <HYPERLINK id="link4" to="AlbumData"/>
</INDEXUNIT>
<DATAUNIT id="AlbumData" entity="Album">
  <INFOLINK id="link5" to="AlbumSupports"/>
  <INFOLINK id="link6" to="ArtistShortData"/>
  <INFOLINK id="link7" to="AlbumTracks"/>
</DATAUNIT>
<MULTIDATAUNIT id="AlbumSupports" entity="AlbumToSupports">
</MULTIDATAUNIT>
<MULTIDATAUNIT id="AlbumTracks" entity="AlbumToTrack">
</MULTIDATAUNIT>
<DATAUNIT id="ArtistShortData" entity="AlbumToArtist">
</DATAUNIT>
<PAGE id="Artists">
  <UNIT id="AllArtists"/>
</PAGE>
<PAGE id="Artist">
  <UNIT id="ArtistData"/>
</PAGE>
<PAGE id="AlbumIndex">
  <UNIT id="ArtistShortData"/>
  <UNIT id="AlbumIndex"/>
</PAGE>
<PAGE id="Album">
  <UNIT id="AlbumData"/>
  <UNIT id="AlbumSupports"/>
  <UNIT id="ArtistShortData"/>
  <UNIT id="AlbumTracks"/>
</PAGE>
</root>
  
```

Figure 7. The .xml file used as input.

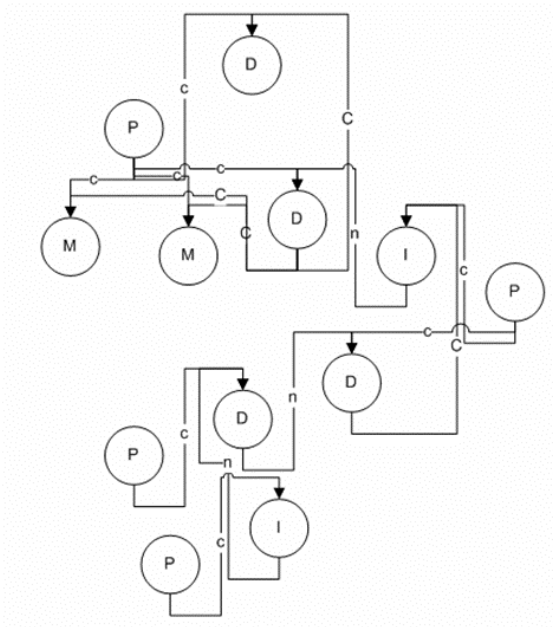


Figure 8. The generated graph representation of the initial .xml file in Visio format.

Table 2. Generated statistics about subgraph occurrences and respective sizes in the conceptual schema of the overall Web application.

Times appearing	Subgraph size	No of subgraphs
4	2	1
3	2	1
3	3	2
3	4	1
2	2	2
2	3	5
2	4	11
2	5	13
2	6	7
2	7	4
1	2	2
1	3	5
1	4	6
1	5	20
1	6	42
1	7	60
1	8	60
1	9	41
1	10	18
1	11	4
TOTAL No. of subgraphs		305

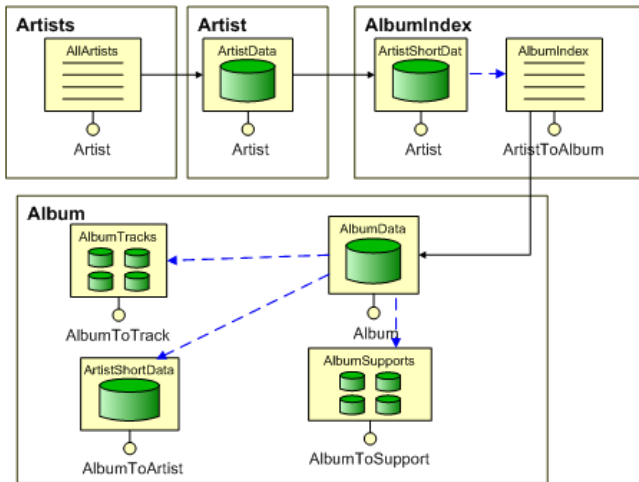


Figure 9. The generated WebML representation of the initial .xml file.

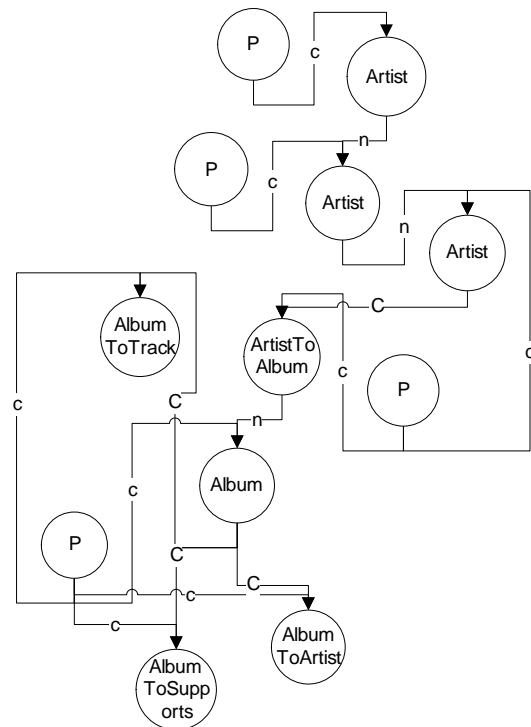


Figure 10. The second generated graph representation of the initial .xml file in Visio format.

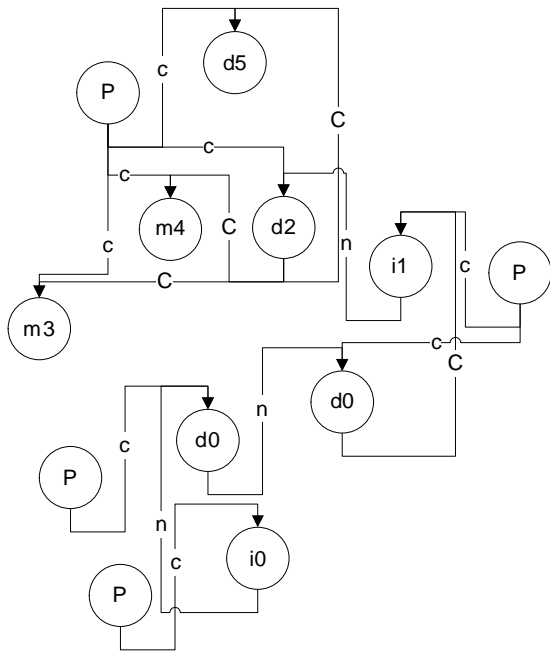


Figure 11. The third generated graph representation of the initial .xml file in Visio format.

Table 3. Generated statistics about subgraph occurrences and respective sizes in in the second case.

Times appearing	Subgraph size	No of subgraphs
112	2	9
85	1	11
75	2	7
62	1	8
57	1	7
52	1	6
48	2	5
44	1	9
42	2	6
40	1	12
36	1	4
33	1	5
16	1	10
15	2	3
15	1	4
6	1	3
5	2	2
4	4	4
3	4	2
3	4	3
3	3	5
1	1	2
TOTAL No. of subgraphs		757

Table 4. Generated statistics about subgraph occurrences and respective sizes in the third case.

Times appearing	Subgraph size	No of subgraphs
2	4	1
TOTAL No. of subgraphs		1

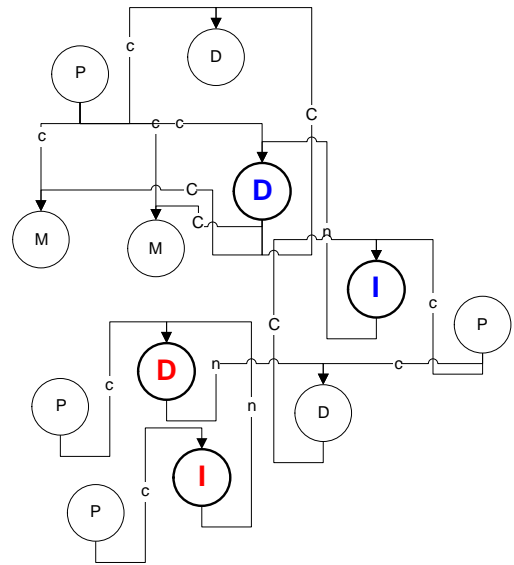


Figure 12. The identified subgraph I-D highlighted.

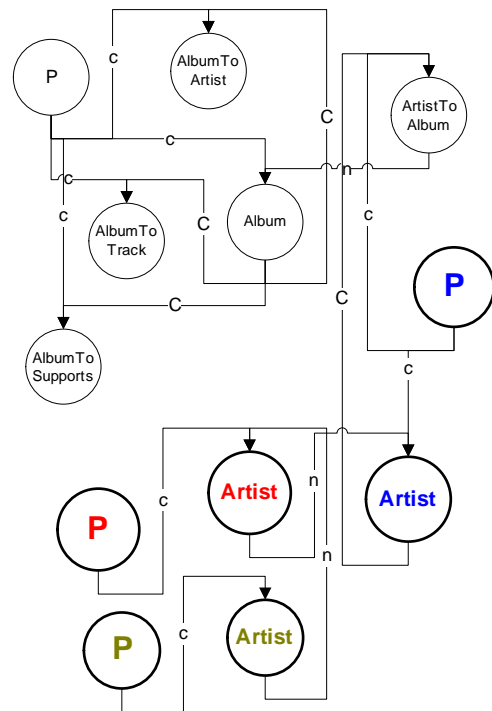


Figure 13. The identified subgraph P-Artist highlighted.

VI. CONCLUSION

In this paper we have illustrated a methodology and a tool that aims at capturing potential problems caused by inappropriate reuse, within the conceptual schema of a Web application. We have introduced the notions of model cloning and model smells, and provided a tool for identifying model clones within an application's hypertext model by mapping the problem to the graph theory domain.

Even though the quality of conceptual schemas highly depends on the selected modeling language, the proposed methodology may be used by a series of languages with minor (and straightforward) adjustments.

The most crucial limitation of the adopted approach is the exhaustive nature of gSpan that locates and examines all potential subgraphs. In the cases where the initial graph is large, the size of the output file of gSpan makes it hard to apply any further manipulation and exploitation. This is the reason why future versions of the tool should embed restrictions concerning the sizes of query subgraphs.

In the future we plan to apply the methodology to a large number of Web application conceptual schemas, in order to refine it and fine-tune the tool. We will also consider the distribution and effect of design patterns within a conceptual schema, in accordance with the process of model clones identification.

REFERENCES

- [1] D.M. Coleman, D. Ash, B. Lowther, & P.W. Oman, "Using Metrics to Evaluate Software System Maintainability", *Computer*, 27(8), 1994, pp. 44-49.
- [2] T. Guimaraes "Managing Application Program Maintenance Expenditure". *Communications of the ACM*, 26(10), 1983, pp. 739-746.
- [3] T. Mens, & T. Tourwe, "A survey of software refactoring", *IEEE Transactions on Software Engineering*, 30(2), 2004, pp. 126-139.
- [4] T. Isakowitz, E.A. Sthor, & P. Balasubramanian, "RMM: a methodology for structured hypermedia design", *Communications of the ACM*, 38(8), 1995, pp. 34-44.
- [5] P. Atzeni, G. Mecca, & P. Merialdo P, "Design and Maintenance of Data-Intensive Web Sites", Proc. 6th International Conference on Extending Database Technology, 1998, pp. 436-450.
- [6] F. Garzotto, P. Paolini, & D. Schwabe, HDM - "A Model-Based Approach to Hypertext Application Design", *ACM Transactions on Information Systems*, 11(1), pp. 1993, 1-26.
- [7] P. Fraternali, & P. Paolini, "A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications", Proc. 6th International Conference on Extending Database Technology, 1998, pp. 421-435.
- [8] D. Schwabe, & G. Rossi, "An object-oriented approach to web-based application design", *Theory and Practice of Object Systems (TAPOS)*, 4(4), 1998, pp. 207-225.
- [9] G. Booch, I. Jacobson, & J. Rumbaugh, *The Unified Modeling Language User Guide*, The Addison-Wesley Object Technology Series, 1998.
- [10] J. Conallen, *Building Web Applications with UML*, Addison-Wesley, 1999.
- [11] J. Conallen, "Modeling Web application architectures with UML", *Communications of the ACM*, 42(10), 1999, pp. 63-70.
- [12] S. Ceri, P. Fraternali, & A. Bongio, "Web Modeling Language (WebML): a Modeling Language for Designing Web Sites", Proc. WWW9 Conference, Amsterdam, 2000.
- [13] B. Boehm, *Software Engineering Economics*, Prentice Hall PTR, 1981.
- [14] E. Chikofsky, & J. Cross, "Reverse engineering and design recovery: A taxonomy", *IEEE Software*, 7(1), 1990, pp. 3-17.
- [15] M. Fowler, K. Beck, J. Brant, W. Opdyke, & D. Roberts, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.
- [16] JH. Johnson "Identifying Redundancy in Source Code using Fingerprints." In the Proceedings of the CAS Conference, Toronto, Canada, 1993, pp. 171-183.
- [17] U. Manber "Finding Similar Files in a Large File System", Proc. of the USENIX Winter 1994 Technical Conference, San Francisco, USA, 1994, pp. 1-10.
- [18] JH Johnson "Substring Matching for Clone Detection and Change Tracking", Proc. of the International Conference on Software Maintenance, Victoria, Canada, 1994, pp. 120-126.
- [19] BS. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems". In the Proceedings of the Second Working Conference on Reverse Engineering, Toronto, Canada, 1995, pp. 86-95.
- [20] Rieger M, Ducasse S Visual Detection of Duplicated Code. In the Proceedings of the Workshop on Experiences in Object-Oriented Re-Engineering, Brussels, Belgium, 1998, pp. 75-76.
- [21] S. Ducasse, M. Rieger, S. Demeyer, "A language independent approach for detecting duplicated code." In the Proceedings of the International Conference on Software Maintenance, IEEE Computer Society Press, Oxford, UK, 1999, pp. 109-118.
- [22] T. Kamiya, S. Kusumoto, & K. Inoue "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code." *IEEE Transactions On Software Engineering* 28(7), 2002, pp. 654-670.
- [23] K. Kontogiannis, "Evaluation Experiments on the Detection of Programming Patterns Using Software Metrics." In the Proceedings of the 4th Working Conference on Reverse Engineering, Amsterdam, The Netherlands, 1997, pp 44-54.
- [24] ID. Baxter, A. Yahin, L. Moura, M. Santa Anna, L. Bier "Clone Detection Using Abstract Syntax Trees." In the Proceedings of the International Conference on Software Maintenance, Washington DC, USA, 1998, pp. 368-377.
- [25] J. Krinke "Identifying Similar Code with Program Dependence Graphs." In the Proceedings of the 8th Working Conference on Reverse Engineering, Stuttgart, Germany, 2001, pp. 301-309.
- [26] J. Mayrand, C. Leblanc, EM. Merlo "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics." In the Proceedings of the International Conference on Software Maintenance, Monterey, USA, 1996, pp 244-254.
- [27] B. Lague, D. Proulx, J. Mayrand, EM. Merlo, J. Hudepohl "Assessing the Benefits of Incorporating Function Clone Detection in a Development Process", Proc. of the International Conference on Software Maintenance, Bari, Italy, 1997, pp. 314-321.
- [28] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, K. Kontogiannis "Measuring Clone Based Reengineering Opportunities", Proc. of the 6th IEEE International Symposium on Software Metrics, Boca Raton, USA, 1999, pp. 292-303.
- [29] G. Antoniou, G. Casazza, M. Di Penta, E. Merlo "Modeling Clones Evolution Through Time Series", Proc. of the International Conference on Software Maintenance, Florence, Italy, 2001, pp. 273-280.
- [30] GA. Di Lucca, M. Di Penta, AR. Fasolino, P. Granato "Clone Analysis in the Web Era: an Approach to Identify Cloned Web Pages", Proc. of the 7th IEEE Workshop on Empirical Studies of Software Maintenance, Florence, Italy, 2001, pp. 107-113.
- [31] C. Boldyreff, R. Kewish "Reverse Engineering to Achieve Maintainable WWW Sites", Proc. of the 8th Working Conference on Reverse Engineering (WCRE'01), Stuttgart, Germany, 2001, pp. 249-257.
- [32] F. Ricca, P. Tonella "Using Clustering to Support the Migration from Static to Dynamic Web Pages." In the Proceedings of the 11th International Workshop on Program Comprehension, Portland, USA, 2003, pp. 207-216.
- [33] M. Rigou, S. Sirmakessis, & G. Tzimas, "Model Cloning: A Push to Reuse or a Disaster?" in *Adaptive and Personalized Semantic Web*: Proc. 16th ACM Hypertext, Springer, Studies in Computational Intelligence (SCI) Vol. 14, pp. 37-55, 2006.
- [34] WebRatio (2006). Available at: <http://www.webratio.com>.
- [35] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, & M. Matera, *Designing Data-Intensive Web Applications*, New York: Morgan Kaufmann, 2002.
- [36] MR Garey, & DS Johnson DS, (1979), *Computers and Intractability: A guide to NP-Completeness*, New York: Freeman, 1979.
- [37] X. Yan, & J. Han, "gSpan: Graph-based substructure pattern mining", Proc. International Conference on Data Mining (ICDM'02), Maebashi, 2002, pp. 721-724.

- [38] X. Yan, & J Han, "CloseGraph: mining closed frequent graph patterns", Proc. of ACM KDD03, 2003, pp 286-295
- [39] C. Wang, W. Wang, J. Pei, Y. Zhu, & B. Shi, "Scalable Mining of Large Disk-based Graph Databases", Proc. of ACM KDD04, 2004, pp 316-325
- [40] R. Giugno, D. Shasha, "GraphGrep: A Fast and Universal Method for Querying Graphs", Proc. International Conference in Pattern recognition (ICPR), Quebec, Canada, August 2002. Available at: <http://www.cs.nyu.edu/shasha/papers/graphgrep/index.html>.

include Web Engineering, Web Modeling, Intranets/Extranets and HCI. He has publications in more than 40 international journals and conference proceedings at these areas.

Evanthia K. Faliagka (Karditsa, 1983) is an undergraduate student in Computer Engineering and Informatics (Computer Engineering and Informatics Department, University of Patras, Greece). Her current research interests include Web Engineering and Web Modeling.

Maria I. Rigou (Athens, 1975) holds a diploma in Computer Engineering and Informatics (Computer Engineering and Informatics Department, University of Patras, Greece, Dec. 1997), an MSc in Computer Science and Technology (Computer Engineering and Informatics Department, University of Patras, Greece, Dec. 2000, thesis: "Interactive Systems Evaluation") and a PhD in Computer Science (Computer Engineering and Informatics Department, University of Patras, Greece, July 2005, dissertation: "Effective Personalization Algorithms based on Web Usage Mining"). Her professional background includes technical coordination of R&D projects with national and EU funding and S/W development for the private sector. She is currently an adjunct assistant Professor at the Computer Engineering and Informatics Department at the University of Patras teaching the course "e-Business" and a scientific collaborator of the Research Academic Computer Technology Institute (Patras, Greece). Her current research interests lay in the areas of Adaptive Hypermedia, Web Personalization and Web Semantics. She has publications in more than 8 journals and more than 30 conference proceedings and is co-author of 5 book chapters and 2 books.

Spiros P. Sirmakessis (Ioannina, 1969) holds a diploma in Computer Engineering and Informatics (Computer Engineering and Informatics Department, University of Patras, Greece, 1992), and a PhD in Computer Science (Computer Engineering and Informatics Department, University of Patras, Greece, 1997, dissertation: "Computational Geometry and Multidimensional Data structures"). He is currently an Assistant Professor at the Department of Applied Informatics in Administration and Economy in the Technological Educational Institution of Messolonghi. Since 2004, he is the Head of the Data Bases and Information Systems Sector and Director of the e-Business and Human Computer Interaction Lab in the same department. During his professional career he has undertaken numerous duties. He cooperates with RACTI since 1992 as the Coordinator of the Internet and Multimedia Research Unit. He has coordinated and participated in several R&D projects since 1992. These projects have been funded by national and European funds. Since 1997, he works in the area of software engineering, web mining, web engineering and distance learning. He is author of 2 books and 6 chapters in books and editor of 3 books. His papers have been published in journals (international and national) and international and national conferences. Updated information is available at <http://www.hci-course.gr/bio.htm>. Since 2001, he is a Consulting Professor in the Hellenic Open University. Prof. Sirmakessis is member of ACM, IEEE, and IASTED. He has chaired several workshops and special sessions (refer to <http://www.hci.gr/events.asp>).

Giannis E. Tzimas (Trikala, 1972) holds a diploma in Computer Engineering and Informatics (Computer Engineering and Informatics Department, University of Patras, Greece, Dec. 1995), and a PhD in Computer Science (Computer Engineering and Informatics Department, University of Patras, Greece, July 2005, dissertation: "Performance Optimization & Effective Design Solutions for Web Applications"). He is currently an adjunct assistant Professor at the Computer Engineering and Informatics Department at the University of Patras teaching the course "Human Computer Interaction" and the technical coordinator of the Internet and Multimedia Technologies Research Unit in Research Academic Computer Technology Institute (<http://www.cti.gr>) (Patras, Greece). He has significant professional experience in Web applications' design and development and he was the project manager of several R&D projects, such as the SOPHIA intranet of the Athens 2004 Organizing Committee. His current research interests