A Structured Approach to Data Reverse Engineering of Web Applications

Roberto De Virgilio and Riccardo Torlone

Università Roma Tre, Italy
{devirgilio,torlone}@dia.uniroma3.it

Abstract. The majority of documents on the Web are written in HTML, constituting a huge amount of legacy data: all documents are formatted for visual purposes only and with different styles due to diverse authorships and goals and this makes the process of retrieval and integration of Web contents difficult to automate. We provide a contribution to the solution of this problem by proposing a structured approach to data reverse engineering of data-intensive Web sites. We focus on data content and on the way in which such content is structured on the Web. We profitably use a Web data model to describe abstract structural features of HTML pages and propose a method for the segmentation of HTML documents in special blocks grouping semantically related Web objects. We have developed a tool based on this method that supports the identification of structure, function, and meaning of data organized in Web object blocks. We demonstrate with this tool the feasibility and effectiveness of our approach over a set of real Web sites.

1 Introduction

With the growth of the Internet, Web applications have become the most important means of electronic communication, especially for commercial enterprisers of all kinds. Unfortunately, many Web applications are poorly documented (or not documented at all) and poorly structured: this makes difficult the maintenance and the evolution of such systems. This aspect, together with the growing demand to reimplement and evolve legacy software systems by means of modern Web technologies, has underscored the need for Reverse Engineering (RE) tools and techniques for the Web. Chikofsky describes RE as "*the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction*" [6]. The Data Reverse Engineering (DRE) emerged from the more general problem of reverse engineering: while RE operates on each of the three main aspects of an information system (data, process, and control), DRE concentrates on data and on its organization. It can be defined as a collection of methods and tools supporting the identification of structure, function, and meaning of data in an software application. In particular, DRE aims at recovering the semantics of the data, by retrieving data structures and constraints, and relies on *structured* techniques to model, analyze, and understand existing

M. Gaedke, M. Grossniklaus, and O. Díaz (Eds.): ICWE 2009, LNCS 5648, pp. 91–105, 2009.

applications of all kinds. It is widely recognized that these techniques can greatly assist for system maintenance, reengineering, extension, migration and integration and motivate the add-on of a framework supporting the complete process of data reverse engineering of Web applications. In this scenario, several approaches have been proposed to convert HTML Web pages into more or less structured formats (e.g. XML or relational tables). Usually, these approaches leverage the structural similarities of pages from large Web sites to automatically derive data *wrappers* (see [12] for a survey). Most of them rely on hierarchy-based algorithms that consider any two elements as belonging to the same item when their corresponding HTML tags are located under a common parent tag in the DOM tree [8,16]. However, when the HTML structure of a Web page became more complicated, an item with several elements can be extracted incorrectly: related elements may be visually positioned closely but textually located under different parent tags in the tree hierarchy. Moreover, the result of the reverse engineering process is a repository of data (e.g., a collection of relational tables) that is poorly processable without user supervision mainly because they take into account the semantics of data only to a limited extent.

In this framework, we propose a structure discovery technique that: (i) identifies blocks grouping semantically related objects occurring in Web pages, and (ii) generates a logical schema of a Web site. The approach is based on a page segmentation process that is inspired by a method to group elements of a Web page in blocks according to a cognitive visual analysis [5]. Visual blocks detection is followed by a pattern discovery technique that generates structural blocks that are represented in a conceptual model, called Web Site Model (WSM) [9]. This model generalizes various (data and object oriented) Web models and allows the representation of the abstract features of HTML pages at content, navigation and presentation levels. Content and presentation are linked to these blocks to produce the final logical schema of the Web site.

An important aspect of our approach is that we face with a highly heterogeneous collection of HTML documents. Specifically, we start from the observation that even if HTML documents are heterogeneous in terms of how topic specific information is represented using HTML markups, usually the documents exhibit certain domain-independent properties. In particular, an HTML document basically presents two types of elements: *block elements* and *text elements*. The former involve the document structure (i.e. headings, ordered/unordered lists, text containers, tables and so on), the latter refer to text inside block elements (e.g., based on font markups). Together, these elements specify information at different levels of abstraction. We distinguish several types of blocks due to their functionality in the page: (i) visual or cognitive, (ii) structural and (iii) Web object. Starting from the visual rendering of a Web page, it is straightforward to divide the page in well-defined and well-identifiable sections according to the cognitive perception of the user. In these *visual blocks* we identify a set of *patterns* that represent structures aggregating information. Each pattern is a collection of tags. For instance the pattern `HTML-BODY-UL-LI` identifies a *structural block* that organizes related information as a list. By grouping patterns, we

identify several *Web object blocks* representing aggregations of information in the page that give hints on the grouping of semantically related objects. Each Web object block represents a particular hypertextual element that organizes and presents a content with a specific layout. We have developed a tool, called ReverseWeb, implementing the above mentioned methods to semi-automatically identify structure, function, and meaning of Web data organized in Web object blocks. ReverseWeb has been used to perform experiments on publically available Web sites.

The paper is structured as follows. In Section 2 we present some related works. In Section 3 we introduce the page segmentation technique to individuate visual and structural blocks. In Section 4 we illustrate how we can identify blocks and produce a logical description of the Web site. In Section 5 we show an architecture of the tool and a number of experimental results and finally, in Section 6, we sketch concluding remarks and future works.

2 Related Work

The literature proposes many methods and tools to analyze Web page structures and layout, with different goals.

UML based approaches. The majority of Web Application reverse engineering methodologies and tools rely on the Unified Modeling Language (UML). UML-based techniques provide a stable, familiar environment to model components as well as the behavior of applications. Among them, Di Lucca et al. have developed the Web Application Reverse Engineering (WARE) tool [10], a very well documented example to RE. The approach is based on the Goals, Models and Tools (GMT) paradigm of Benedussi and makes use of the Conallen UML extensions to represent information as package diagrams (use-case diagrams for functional information, class-diagrams for the structure, and sequence-diagrams for the dynamic interaction with the Web Application). Chung and Lee [7] also adopt the Conallen extensions. They represent the Web content in terms of a component diagram and the Web application in terms of a package diagram. In general, all of these approaches focus on the behavior and interaction with a Web Application, rather than on its organization.

Ontology based approaches. The basic idea of these approaches is to model a Web application by means of an schema. Among them, Benslimane et. al [3] and Bouchiha et. al. [4] have proposed OntoWare, whose main objective is the generation of an ontological, conceptual representation of the application. The authors criticize other approaches to reverse engineering because they do not provide adequate support to knowledge representation (a position also supported by Du Bois [11]). The ontological approach provides a high level analysis of a Web Application but usually depends on the specific domain of interest. In most cases, data extraction can only be done after a user intervention aimed at building the domain ontology by locating and naming Web information [2].

Usually, the ontology based approaches rely on a specific formalism to represent the structures extracted from Web pages. Lixto [2] is a tool for the generation of wrappers for HTML and XML documents. Patterns discovered Lixto are here expressed in terms of a logic-based declarative language called Elog.

Source code based approaches. Ricca and Tonella have proposed ReWeb [13], a tool for source code analysis of Web Applications. They use a graph model to represent a Web application and focus on reachability, flow and traversal analysis. The outcome of the analysis is a set of popup windows illustrating the evolution of the Web Application. Vanderdonckt et al. [15] have developed VAQUISTA, a framework to reverse engineering the interface of Web applications. The aim of this work is to facilitate the migration of Web Application between different platforms. VAQUISTA performs a static analysis of HTML pages and translates them into a model describing the elements of the HTML page at different levels of abstraction. Antoniol et. al. [1] use an RMM based methodology. The authors apply an RE process to identify logical links which are then used to build a Relationship Management Data Model (RMDM). From the RMDM an Entity-Relationship model is then abstracted. This is the end point of the reverse engineering process. In general, all of these solutions produce a logical description of a specific aspect of a Web application (mainly related to the presentation).

3 Extraction of Page Structure

3.1 Overview

Our approach is related to recent techniques for extracting information from the Web [12]. As for most of these proposals, we start from the observation that data published in the pages of large sites usually (i) come from a back-end database and (ii) are embedded within shared HTML templates. Therefore the extraction process can rely on the inference of a description of the shared templates. Though this approach is applicable on Web documents, it does not exploit the hypertext structure of Web documents. Our work focuses on discovering this structure as well. Some research efforts show that users always expect that certain functional part of a Web page (e.g., navigational links, advertisement bar and so on) appears at certain position of a page[1]. Additionally, there exist blocks of information that involve frequent HTML elements and have a higher coherence. That it to say, in Web pages there are many unique information features, which can be used to help the extraction of blocks involving homogeneous information.

To this aim we define a Data Reverse Engineering (DRE) process composed by the following steps:

- *Page Segmentation*: each Web page in a Web site is segmented in several blocks according to the visual perception of a user. Each resulting visual block of a Web page is isolated and through an analysis of the DOM associated with blocks, a set of structural patterns are derived. Differently from

[1] For more details see `http://www.surl.org/`

other approaches, this step combines a computer vision approach (to understand the perception of users) with a DOM structure extraction technique (which conveys the intention of Web authors).
- *Schema Discovery*: the derived patterns of each Web page, grouped in visual blocks, suggest the clustering of pages in the Web site. Each cluster represents aggregations of semantically related data and is represented by a set of structural patterns. Usually, a wrapper is generated to automate the extraction of patterns and to structure the Web content associated with them according to a logical model (e.g, the relational model). Differently from these solutions, in this step we make use of a conceptual model to represent Web data at content, navigation and presentation levels. The patterns of each cluster are mapped into constructs of this model. Finally, based on this conceptual representation, a logical schema of the Web site is extracted.

In the rest of this section, we will describe in detail the page segmentation phase, providing algorithms to compute the structural patterns. In the next section we will illustrate the schema discovery technique.

3.2 Page Segmentation

We start by exploiting the semi-structured nature of Web documents described in terms of their document object model (DOM) representation. Note that DOM poorly reflects the actual semantic structure of a page. However the visual page layout structuring is more suitable to suggest a semantic partitioning of a page. Therefore, we make use of the VIPS approach (Vision-based Page Segmentation) [5], taking advantage from DOM trees and visual cues. The main idea is that: (i) semantically related contents are often grouped together, and (ii) the page usually divides the content by using visual separators (such as images, lines, and font sizes). VIPS exploits the DOM structure and the visual cues and extracts information blocks according to the visual perception. The output of VIPS associates with each is Web page a partitioning tree structure over *visual blocks* (VBs). The resulting VBs present a high *degree of coherence*, meaning that they convey homogeneous information within the page. Then we assign an XML description to the tree: each VB is identified by the path from the root of the page in its DOM representation (considering also the available styling information of `class` or `id` referring to the associated Cascading Style Sheet or CSS) and is characterized by the position in the page. For instance, Figure 1 shows the resulting partitioning tree and XML description of the home page of Ebay[2].

Let us consider the visual block VB2_1 of Figure 1: it organizes information using an unordered list of items. Figure 2 shows an extract of DOM and CSS properties for VB2_1. The next phase analyzes each identified VB and discovers repeated patterns representing aggregations of information with a shared structure. More in detail, in each VB we label any path from the root of VB to a node using an hash code. A preorder traversal generates a sequence V representing a vector of hash codes, as shown in Figure 2.

[2] `http:\www.ebay.com`

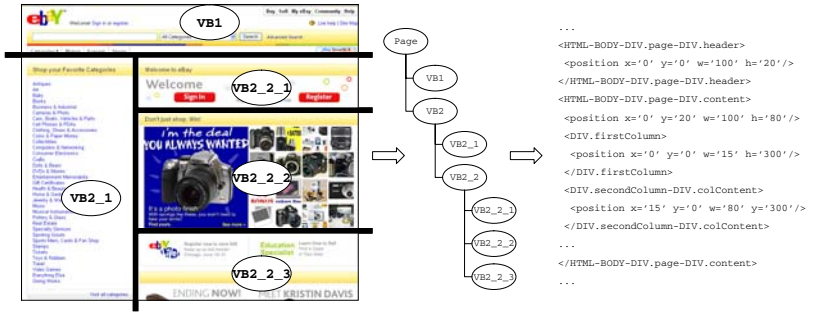

Fig. 1. Visual Partitioning of a Web page

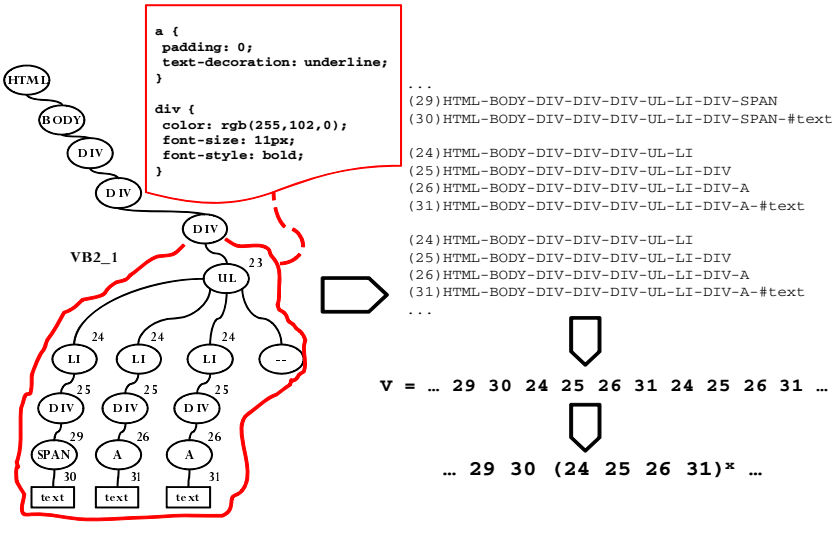

Fig. 2. Pattern searching

Finally we group repeated paths that represent patterns to identify. To this aim we use an algorithm, called *path-mark* that gets inspiration from the dictionary-based compression algorithm LZW [14]. Algorithm 1. illustrates the pseudo-code of path-mark.

The algorithm manages a queue Q and a sequence V, and returns a map M where each group of hash codes has assigned the number of its occurrences in V. We generate V, initialize Q and M (lines $4-5$) and make use of a window (*win*) to scan the subsequences to analyze (line 7). *win* varies from one to half of the length of V. So we extract a candidate subsequence (*actual*) of *win* length and insert it in Q (lines $9-11$). We compare *actual* with the top subsequence (*previous*) in Q, previously analyzed, and if they are equal we count the number of consecutive occurrences (*counter*) of *actual* in the rest of V moving with a *win* scale. At the end we assign *counter* to *actual* in M (lines $12-22$). Otherwise we extract another subsequence and iterate the algorithm from the line 8. Referring

Algorithm 1. Path-Mark

```
1: Input: A visual block VB
2: Output: A Map of occurring patterns in VB, each one related to its occurrences
3: begin
4: V ← HASHPREORDER(VB) // V is a sequence
5: EMPTY(M), EMPTY(Q) // M is a map and Q a queue
6: seq_length ← LENGTH(V)
7: for win from 1 to seq_length/2 do
8:     for index from 0 to seq_length − win do
9:         previous ← DEQUEUE(Q)
10:        actual ← SUBSEQUENCE(V, index, index + win)
11:        ENQUEUE(Q, actual)
12:        if actual = previous then
13:            counter ← counter + 1
14:            internal ← index + win
15:            while internal < seq_length do
16:                next ← SUBSEQUENCE(V, internal, internal + win)
17:                if actual = next then counter ← counter + 1
18:                elseINSERT(M, actual, counter)
19:                end if
20:                internal ← internal + win
21:            end while
22:        end if
23:    end for
24: end for
25: return M
26: end
```

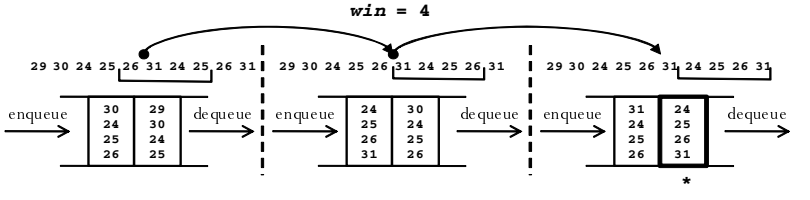

Fig. 3. An execution of Path-Mark

to the example of Figure 2, we show an execution of the algorithm in Figure 3 with $win = 4$.

Our algorithm returns a grouping such as: $(\ldots, 29, 30, (24, 25, 26, 31)^x, \ldots)$. This means that $(24, 25, 26, 31)$ is a repeated path `HTML-BODY-DIV-DIV-DIV-UL-LI-DIV-A` that presents a pattern `UL-LI-DIV-A`, where `HTML-BODY-DIV-DIV-DIV` is the root of the container VB2_1.

4 Schema Discovery

In the previous section we have presented a technique to segment a Web page into structural blocks representing aggregations of semantically related data. The following step consists of generating a logical schema matching the discovered patterns making use of the Web Site Model described in [9].

To this purpose, we get inspiration from the idea of Crescenzi et al. [8]: a Web page p can be considered as a couple $\{ID, VB\}$, where ID is an identifier and VB is the set of visual blocks, resulting by the VIPS segmentation. Each VB_i is

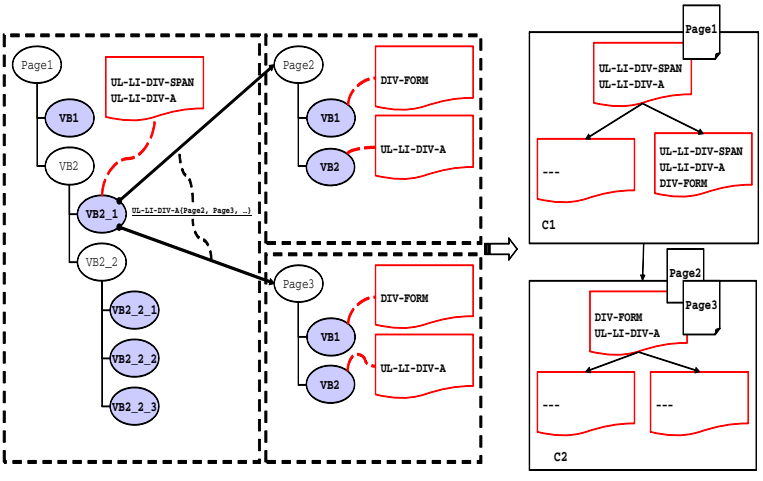

Fig. 4. From pages to clusters

a collection of patterns $pt_1, pt_2, \ldots$, identified by the path-mark algorithm shown in the previous section. So we define the *page schema* of a Web page p as the union of all patterns occurring in each visual block. Then we call *link collection* in a Web page p all node-to-link patterns together with all the URLs that share that pattern. For instance consider the Web pages *Page1*, *Page2* and *Page3* in left side of Figure 4. They are described in terms of a tree of visual blocks. Each block is associated with a set of patterns, identified by Algorithm 1. Both *Page2* and *Page3* have a page schema described by the set {`DIV-FORM, UL-LI-DIV-A`} and the link collection {`UL-LI-DIV-A`{url_1, url_2, ...}}. *Page1* is associated with the link collection {`UL-LI-DIV-A`{*Page*2, *Page*3, ...}}.

We can establish a partial ordering between page schemas by introducing the notions of *subsumption* and *distance*. Given two page schemas ps_1 and ps_2, we say that ps_1 is subsumed by ps_2, $ps_1 \lhd ps_2$, if each pattern pt in ps_1 also occurs in ps_2. The distance between two page schemas is defined as the normalized cardinality of the symmetric set difference between the two schemas. Let us consider again ps_1 and ps_2, then $dist(ps_1, ps_2) = \frac{|(ps_1 - ps_2) \cup (ps_2 - ps_1)|}{|ps_1 \cup ps_2|}$. Note that if $ps_1 = ps_2$ (that is, the schemas coincide), then $dist(ps_1, ps_2) = 0$. If $ps_1 \cap ps_2 = \emptyset$ (the schemas are disjoint), then $dist(ps_1, ps_2) = 1$. Based on the notions of page schema, subsumption and distance we then define a notion of *cluster* as a collection of page schemas: a cluster is a tree $\{N_C, E_C, r_C\}$ where (i) N_C is a set of nodes representing page schemas, (ii) E_C is a set of edges (n_i, n_j) such that $n_i \lhd n_j$, and (iii) r_C is the root. In a cluster, a page schemas ps_i is parent of a page schema ps_j if $ps_i \lhd ps_j$, therefore the root of a cluster represents the most general page schema in the cluster. Each page schema is associated with a set of Web pages that match with it. To maintain clusters we use a threshold dt. Given a cluster C and a page schema ps and the set of associated pages, ps

can be inserted in C if $dist(ps, r_C)$ is lower than the given threshold dt[3]. For instance, in right side of Figure 4 there are the clustering of *Page1*, *Page2* and *Page3*. Now we can define also a notion of *cluster link*: given a cluster C_1 and one of its pattern node-to-link pt, consider the link collections of the Web pages in C associated with pt. We say that there exists a cluster link L between C_1 and the cluster C_2 if there are links in the link collections associated to pt that point to pages in C_2.

A relevant step in schema discovery is the computation of a useful partition of Web pages in clusters, such that pages in the same cluster are structurally homogeneous. Whereupon a crawler navigates a Web site starting from the home page and an agglomerative clustering algorithm groups pages into classes. We have designed an algorithm that builds a set of clusters incrementally. Algorithm 2. shows the pseudo code.

Algorithm 2. Compute Clusters

Require: n: max size of selected links subset
Require: dt: distance threshold for candidate selection
1: **Input:** Starting Web page p_0
2: **Output:** the set of Clusters CL
3: **begin**
4: $EMPTY(CL), EMPTY(Q)$ // *CL is a set and Q a queue*
5: $INSERT(p_0, CL, dt)$
6: $Q \leftarrow LINKCOLLECTION(p_0)$
7: **while** Q *is not empty* **do**
8: $lc \leftarrow DEQUEUE(Q)$
9: $W \leftarrow PAGES(lc, n)$
10: $H \leftarrow \emptyset$
11: **while** W *is not empty* **do**
12: $W - \{p\}$
13: $INSERT(p, CL, dt)$
14: $H \cup LINKCOLLECTION(p)$
15: **end while**
16: **while** H *is not empty* **do**
17: $H - \{lc'\}$
18: $ENQUEUE(Q, lc')$
19: **end while**
20: **end while**
21: **return** CL
22: **end**

The input of the algorithm is the home page p_0 of the Web site, which is the first member of the first cluster in the set CL (line 5). The output is the set of computed clusters CL. From p_0 we extract its link collections, and push them into a priority queue Q (line 6). Then, the algorithm iterates until the queue is empty. At each iteration a link collection lc is extracted from Q (line 8), and a subset W of the pages (n) pointed by its links is fetched (line 9)[4]. Then the pages in W are grouped according to their schemas (lines 11-15). The function $INSERT(p, CL, dt)$ inserts a page p into a cluster of CL all the pages whose page schema has a distance from the root r_C lower than the threshold dt. Basically, we extract the page schema ps of p, by using the path-mark algorithm, and select

[3] On the basis of our experiments, we have set $dt = 0.4$.
[4] We assume that is sufficient to follow a subset of the potentially large set of links to determine the properties of the entire collection.

the cluster C in CL whose root has the minimum distance from ps (lower than dt). If there is no cluster satisfying these properties, we add to CL a new cluster having ps as root.

Starting from the root of C, we insert ps (and p) into C as follows: (i) if there is no child n of the root r_C of C such that $n \lhd ps$, then (a) ps becomes the child of r_C, and (b) each child n of r_C such that $p \lhd n$ becomes child of ps; (ii) otherwise, we insert ps in the sub-tree of C having as root the child n of r_C such that (a) $n \lhd ps$, and (b) the distance between ps and n is minimum. Once ps has be inserted in C, we move each n'' such that (i) $ps \lhd n''$ and (ii) n'' is at the same level of ps, as a child of ps.

Then, we extract the link collections of p and update the queue Q (lines 16-19). In this process we assume that the links that belong to the same link collection lead to pages that are similar in structure or with minor differences in their schemas. Then we assign a priority to link collections by visiting the fewest possible pages: an higher priority is given to link collections that have many instances of outgoing links from the cluster. This means that long lists in a page are likely to point pages with similar content (this is particularly true when they are generated by a program), and therefore the next set of pages will provide an high support to discover another cluster.

The final step of the schema discovery process consists of representing each cluster according to our Web Site Model (WSM) [9]. The idea is to identify a set of *container tags* representing candidates to be mapped. In particular we refer to HTML tags that bring to information content in a Web page such as `UL,TABLE,DIV,BODY,`.... Each pattern rooted in a tag container will be translated into a metacontainer using a particular navigational structure (Index, Guided Tour or Entry). We fix a set of heuristics for each construct of our model. Referring to the example of Figure 2 we map the pattern `UL-LI-DIV-A` into an Inde because we have a heuristic that maps a pattern `UL-LI-#-A` with an Index. Each metacontainer is identified by the path from the root to the container tag in the DOM and presents several properties representing the occurring patterns into the block. Then, we organize the data content according to the information content associated to each pattern, and the presentation according to the style properties associated in the Cascading Style Sheet, organized then in WOTs. The root of each cluster is the representative page schema to describe in WSM. As a example, Figure 5 shows the Web object blocks associated with the root of cluster C_1 shown in Figure 4 and the corresponding implementation in a relational DBMS.

5 Experimental Results

On the basis of the methodologies and techniques above described, we have designed a tool for data reverse engineering of data intensive Web applications called ReverseWeb. Figure 6 shows the architecture of the tool.

The main modules of the tool are¿ (i) a *PreProcessor (PP)* and (ii) a *Semantic Engine (SE)*. The PP module is responsible to communicate with the

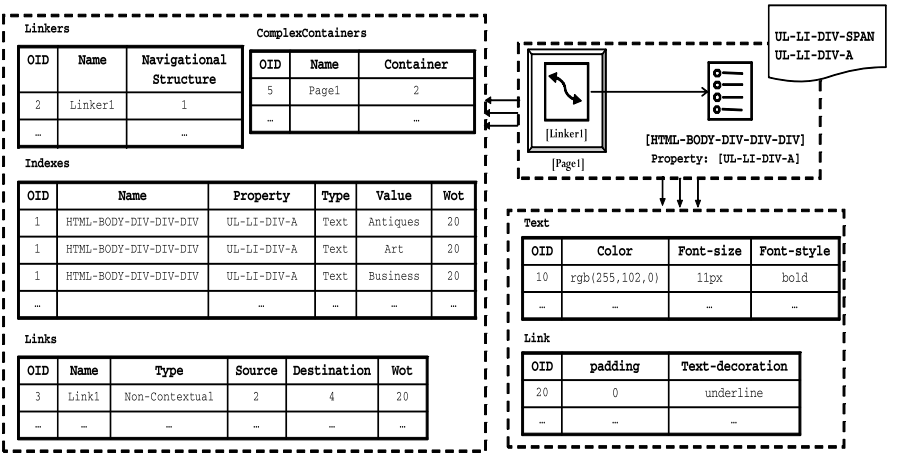

Fig. 5. An example of Web object blocks

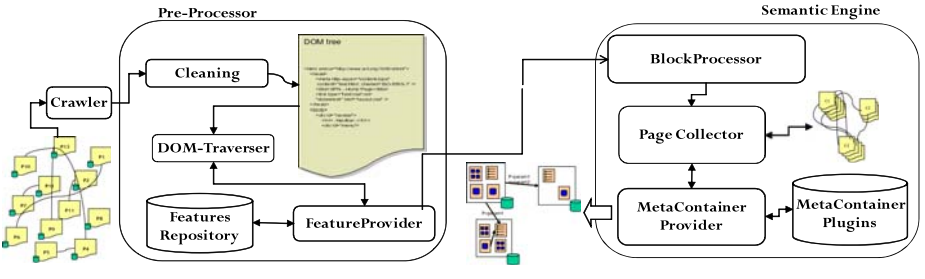

Fig. 6. The Architecture of ReverseWeb

crawler, to process a Web page by using a cleaner (`Tidy` available at `http://tidy.sourceforge.net/`) and a *DOM traverser*, and to produce a structural segmentation of the page. This segmentation is supported by a *Feature Provider* that selects in a repository the segmentation feature to apply (i.e. VIPS and Path-Mark). This choice makes the segmentation step modular and extensible. The resulting XML description (as shown in Section 3.2) is taken as input by the SE module that is responsible to map discovered patterns to metaconstructs of our Web Site Model. The *Block Processor* supports the *Page Collector* to produce and manage the clusters of pages. The resulting set of clusters are taken as input by the *MetaContainer Provider* component that processes the representative page schemas, maps single pattern to a construct by using a repository of *Plugins*, containing the different heuristics, and returns the final logical schema.

ReverseWeb has a Java implementation. The crawling is multi-threading makes use of an internal browser. The GUI has been realized with the SWT toolkit[5]), which has been designed to provide an efficient and portable access

[5] `http://www.eclipse.org/swt/`

Table 1. Experimental Results on 1000 pages

	DIA	EBAY	BUY	WORD	NBA
R .total (sec)	1050,80	2209,39	4589,95	1045,51	1232,68
R .avg (sec)	1,05	2,30	7,57	1,06	1,25
Page dim	471	1108	2054	477	1827
BF dim	179	631	1047	171	804
DRE quality	0,38	0,57	0,51	0,36	0,44

to the user-interface facilities provided by the operating system on which it is implemented, and the NetBeans Visual Graph Library[6]. All algorithms and heuristics have been implemented in Java. The CSS steady state Library[7] has been used to parse the presentation properties of a page.[8]

Plenty of experiments have been done to evaluate the performance of our framework using an Apple computer xServer, equipped with an Intel Core 2 Duo 1.86 Ghz processor, a 4 GB RAM, and a 500 GB HDD Serial ATA. These experiments rely on crawling 1000 pages and producing the logical page schemas of the following Web sites:

1. DIA, the Department of Informatics and Automation of Roma Tre University (`http://web.dia.uniroma3.it/`), and WORD, the Dictionary translator Web Site (`http://www.wordreference.com`);
2. BUY (`http://www.buy.com`) and EBAY (`http://www.ebay.com`), two famous e-commerce Web sites;
3. NBA (`http://www.nba.com`), a well known basketball Web site.

We measured the *average elapsed time* to produce a logical schema and the *accuracy* of the result. In the table 1, for each Web site we show (i) the real time in seconds (R. total) to produce a logical schema, (ii) the average time in seconds (R. avg) to reverse a Web page, (iii) the average page dimension (Page dim) in terms of number of nodes in the DOM, (iv) the average amount of nodes in the DOM of the Web page, involved in the Web object blocks (correctly computed) of the final page schema (BF dim) and (v) the *DRE quality*. The DRE quality measures the accuracy to determine a correct set of Web object blocks as follows. We have adopted the following performance measure: $P_r = \frac{Page_{dim} - BF_{dim}}{Page_{dim}}$ where $Page_{dim}$ is the retrieved portion of a Web site and BF_{dim} is the relevant portion. Basically, P_r is the fraction of the Web site portion retrieved that is not relevant to the schema information need. In other words P_r is the fraction of the Web site portion containing Web content that user will not query.

Then, we define the DRE quality results as: $DRE_{quality} = 1 - P_r$. This coefficient measures the effectiveness of the resulting logical schema. It compares the average amount of nodes involved in the final schema with the average number

[6] `http://graph.netbeans.org/`

[7] `http://cssparser.sourceforge.net/`

[8] More details on ReverseWeb can be found at `http://mais.dia.uniroma3.it/ReverseWeb`, where an alpha version of the tool is publically available.

of nodes for page. More specifically, the percentage of DOM nodes in a page involved in the final schema. This coefficient is in a range [0,1]. If DRE quality is too close to zero, this means that the system was not able to identify significant blocks. Conversely, if the DRE quality is too close to one, the system had difficulties to prune unmeaningly blocks. We have experimentally determined that the best values of DRE quality are in the range [0.2,0.6].

The table provides interesting information about the structure of the analyzed Web sites. DIA, WORD and NBA present the lowest values of R. total. This is a consequence of the regular structure and homogeneity of information blocks in the pages. Moreover they present an optimal DRE quality. EBAY and BUY have higher elapsed times, due to their irregular structure of pages, relevant heterogeneity of published information and great amount of non informative nodes (e.g. banner, spots, and so on), typical in e-commerce Web sites. These results are supported also by diagrams: Figure 7 illustrates the number of Web object blocks and the average elapsed time with respect to the increasing number of DOM nodes in a Web page for the Web sites BUY and NBA. They underline the effectiveness and the add-on of our framework. In Figure 7 is shown the trend of the number of Web object blocks with respect to the increasing number of page nodes. NBA presents an average of 5 blocks for page. This implies that the Web site presents a regular (and complex) structure. The regularity of the site is due to common structure of the published information (regarding basketball teams) and this is close to the reality. This regularity is also supported by the stable average of the elapsed time, shown in Figure 7. Conversely, BUY presents a variable structure of pages with an increasing trend of Web object blocks and

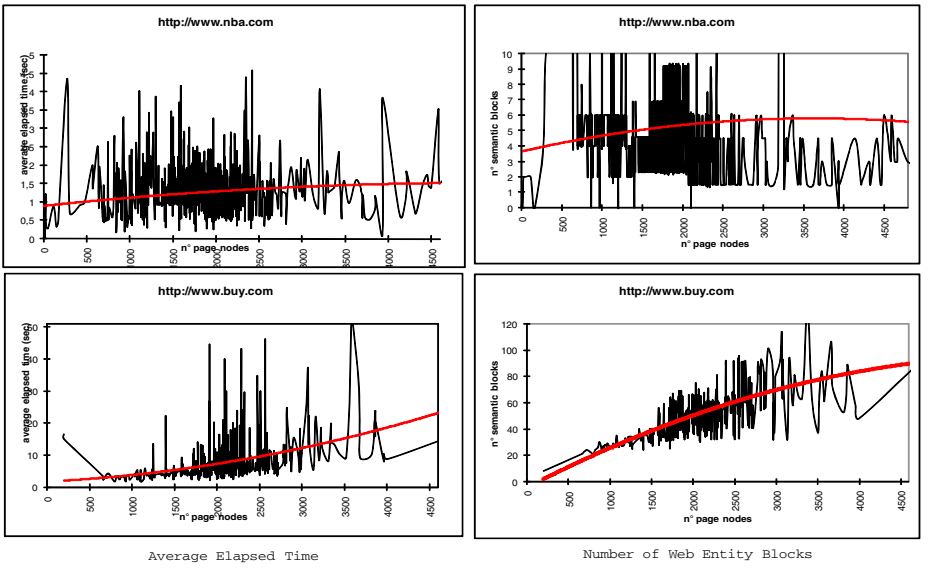

Fig. 7. Average elapsed time and Number of Web object blocks

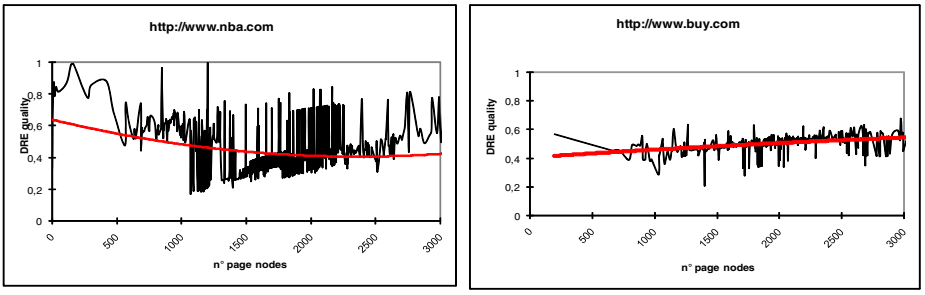

Fig. 8. DRE quality

elapsed times. This is due to the different structure of published information with (i.e., very different products such as art, Hi-Tech and so on). The DRE quality is very good in all Web sites. It presents the best values for DIA, WORD and NBA, over which ReverseWeb worked linearly. In Figure 8 we present the trend of DRE quality with respect to the increasing number of visited nodes. NBA starts with high values, due to the initial computation of clusters. However, as the number of visited nodes increases, the performance improves and converges to an average of 0,44. BUY has an average of 0,51. In summary, plenty of experiments have confirmed the effectiveness of our framework to detect the organization of a Web site.

6 Conclusions and Future Work

In this paper we have addressed the issue to Data Reverse Engineering (DRE) of data-intensive Web Applications. DRE evolved from the more generic reverse engineering process, concentrating on the data of the application and on its organization. We have presented an approach to the identification of structure, function, and meaning of data in a Web site. The approach relies on a number of structured techniques (such as page segmentation) and model-based methods aimed at building a conceptual representation of the existing applications. Moreover, we have evaluated the effectiveness of our approach by implementing a tool, called ReverseWeb, and facing several experiments on different Web sites.

There are several interesting future directions. We are currently trying to improve the segmentation step, by introducing new features in the preprocessing phase. We intend to introduce a notion of polymorphism to optimize the mapping between patterns and Web object blocks. Finally we plan to refine the clustering technique by introducing a distance notion between pages and exploiting this information in the segmentation phase.

References

1. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A.: Web Site Reengineering using RMM. In: Proc. of Int. Workshop on Web Site Evolution, Zurich, Switzerland (2000)

2. Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto. In: Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB 2007), Roma, Italy (2001)
3. Benslimane, S.M., Benslimane, D., Malki, M., Amghar, Y., Hassane, H.S.: Acquiring owl ontologies from data-intensive web sites. In: Proc. of Int. Conf. on Web Engineering (ICWE 2006), Palo Alto, California, USA (2006)
4. Bouchiha, D., Malki, M., Benslimane, S.M.: Ontology based Web Application Reverse Engineering Approach. INFOCOMP Journal of Computer Science 6(1), 37–46 (2007)
5. Cai, D., Yu, S., Wen, J.R., Ma, W.Y.: Extracting Content Structure for Web Pages based on Visual Representation. In: Zhou, X., Zhang, Y., Orlowska, M.E. (eds.) APWeb 2003. LNCS, vol. 2642, pp. 406–417. Springer, Heidelberg (2003)
6. Chikofsky, E.J., Cross, J.H.: Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software 7(1), 13–17 (1990)
7. Chung, S., Lee, Y.S.: Reverse Software Engineering with UML for Web Site Maintenance. In: Proc. of the 1th Int. Conf. on Web Information Systems Engineering (WISE 2000), Hong Kong, China (2000)
8. Crescenzi, V., Merialdo, P., Missier, P.: Clustering Web pages based on their structure. Data Knowl. Eng. 54(3), 279–299 (2005)
9. De Virgilio, R., Torlone, R.: A Meta-model Approach to the Management of Hypertexts in Web Information Systems. In: ER Workshops (WISM 2008) (2008)
10. Di Lucca, G.A., Fasolino, A.R., Tramontana, P.: Reverse engineering Web applications: the WARE approach. Journal of Software Maintenance 16(1-2), 71–101 (2004)
11. Du Bois, B.: Towards a Reverse Engineering Ontology. In: Proc. of the 2th Int. Workshop on Empirical Studies in Reverse Engineering (WESRE 2006), Benevento, Italy (2006)
12. Laender, A., Ribeiro-Neto, B., Da Silva, A., Teixeira, J.S.: A brief survey of web data extraction tools. ACM SIGMOD Record 31(2), 84–93 (2002)
13. Ricca, F., Tonella, P.: Understanding and Restructuring Web Sites with ReWeb. IEEE Multimedia 8(2), 40–51 (2001)
14. Tao, T., Mukherjee, A.: LZW Based Compressed Pattern Matching. In: Proc. of the 14th Data Compression Conf (DCC 2004), Snowbird, UT, USA (2004)
15. Vanderdonckt, J., Bouillon, L., Souchon, N.: Flexible reverse engineering of Web Pages with VAQUISTA. In: Proc. of the 8th Working Conf. on Reverse Engineering (WCRE 2001), Stuttgart, Germany (2001)
16. Wong, T.-L., Lam, W.: Adapting web information extraction knowledge via mining site-invariant and site-dependent features. ACM Transactions on Internet Technology 7(1), 6 (2007)