UNIVERSITÀ DEGLI STUDI DI ROMA TRE
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

Segmentation of HTML Web sites in *Web Entity Blocks*

ROBERTO DE VIRGILIO, RICCARDO TORLONE

RT-DIA-137-2008 **September 2008**

Dipartimento di Informatica e Automazione,
Università di Roma Tre,
Rome, Italy.
{dvr,torlone}@dia.uniroma3.it

ABSTRACT

Data Reverse Engineering is a rapidly growing field of research to make possible the evolution towards the Web 2.0. In this scenario, Web content should be self-descriptive to be automatically interpreted and, possibly, used differently from their original goal. 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 authorship and goals of the people writing these documents. This makes the process of retrieval and integration of Web content difficult to automate. This paper proposes a structured approach to data reverse engineering of data-intensive HTML Web sites. We focus on data content and on the way in which such content is structured on the Web. We first present a Web site data model to describe abstract structural features of HTML pages. Then we show how such model can be profitably used to segment HTML documents in special blocks (*Web entity blocks*) grouping semantically related objects. A framework was developed using methods and tools supporting the identification of structure, function, and meaning of data organized in Web entity blocks. We demonstrate with this framework the feasibility and effectiveness of our approach over a set of real-life 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 [16]. This aspect, together with the growing demand to reevaluate and reimplement legacy software systems by means of World Wide Web technologies, has underscored the need for Reverse Engineering (RE) tools and techniques for the Web. In [13] 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*". The Data Reverse Engineering (DRE) emerged from the more general problem of reverse engineering: while RE can operate on each of the three principal aspects of an information system that are 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 [12]. In particular, DRE aims at recovering the precise semantics of the data, by retrieving data, structures and constraints [20], and relies on *structured* techniques to model, analyze, and understand existing applications of all kinds [1]: 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 (semi) 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 [22] for a survey). However, most of these approaches takes into account the semantics of data only partially. Usually, 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. Other proposals take into account the inherent tree structure of HTML documents [10, 11] but they are merely based on DOM trees [21]. These however focus on presentation and layout and therefore semantically related data that are far from each other in such structures may be processed wrongly.

In this framework, we propose in this paper an approach to Data Reverse Engineering of Web Applications that provides two main contributions. First, it introduces a conceptual model, called Web Site Model (WSM), that generalizes various (data oriented and object oriented) Web models [23] and allows the representation of the abstract features of HTML pages at content, navigation and presentation levels. Moreover, we propose a schema 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 technique relies 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 [8]. Visual blocks detection is followed by a pattern discovery technique that generates structural blocks that will be represented in WSM. Content and presentation are linked to these final 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.

In more detail, 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. Actually, 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 markup tags). 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 entity*. 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 entity blocks* representing aggregations of information in the page, giving hints on the grouping of semantically related objects. Each Web entity block represents a particular hypertext modeling element that organizes a content and presents it with a particular layout.

We have developed a framework, called *ReverseWeb*, implementing the above mentioned methods and tools to semi-automatically identify structure, function, and meaning of Web pages data organized in Web entity blocks. ReverseWeb has been used to perform experiments on real-life 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 the WSM model and show how it can be used to identify blocks and produce a logical description of the Web site. In Section 5 we show an architecture of the tool and 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. However, goals aimed at are various and sometimes totally differ from ours.

UML based methodologies The majority of reported Web Application reverse engineering methodologies and tools are founded on the Unified Modelling Language (UML). UML-based technique provides a stable, familiar environment to model components as well as the behavior of an application. Examples of UML-based methods can be found in the works of Di Lucca et. al. [18], and Chung and Lee [14]. Di Lucca et al. developed the Web Application Reverse Engineering (WARE) tool [17]. It is a very well documented example of RE. This methodology is based on the Goals, Models and Tools (GMT) paradigm of Benedussi [4] and makes use of the UML extensions, defined by Conallen [15] to extract information as package diagrams (i.e. use-case diagrams for functional information, class-diagrams to describe the structure, and sequence-diagrams to document the dynamic interaction with the Web Application). Chung and Lee adopt the Conallen extensions too and consider each Web page as a component, resulting a component diagram, and reflect the Web Application structure as package diagrams. All of these approaches focus on the documentation of behavior and interaction with Web Applications, rather than the organization itself.

Ontology based methodologies The ontology approach to reverse engineering provides a common set of concepts. The focal point of this method is that it models a Web Application

through a schema. Benslimane et. al [5] and Bouchiha et. al. [7] propose OntoWare. Its objective is to generate conceptual schemes. The work of the authors criticize current research contributions on reverse engineering that do not provide adequate knowledge and support (a position supported by Du Bois [19] too). The ontological approach provides an high-level analysis of a Web Application. However it depends on the particular domain of interest. In most cases, data extraction can only be done after a user-oriented step that consists of building the domain ontology by locating and naming the Web information [3, 25]. These different approaches use a specific inner formalism to specify the extraction of structures from HTML pages. Lixto [3] is a wrapper generation tool which is wellsuited for building HTML/XML wrappers. Patterns discovered by the user are internally expressed by a logic-based declarative language called Elog.

Source code based methodologies Ricca and Tonella propose ReWeb [24], a tool to traditional source code analysis of Web Applications. They use a graph structure to represent a Web Application and focus on reachability, flow and traversal analysis. The outcome of the analysis is a set of popup windows to illustrate the evolution of the Web Application using a color coding. Vanderdonckt et al. [27] define VAQUISTA, a framework to reverse engineering the user interface of Web Applications. The aim of the work is to facilitate the Web Application migration to different platforms. VAQUISTA processes a static analysis of HTML pages and translates them into a presentation model describing the elements of the HTML page at different levels of abstraction. Antoniol et. al. [2] use an RMM based methodology. The authors present a university module as an example of Web Application and apply an RE process to identify links which are then used to build a Relationship Management Data Model (RMDM). From the RMDM, and further analysis, an Entity-Relationship model is abstracted. This is the end point for the reverse engineering. All of these solutions produce logical description of a Web Application, however concentrating on specific data aspects (i.e. presentation mainly).

3 Extraction of page structure

3.1 Overview

Our approach is related to recent research for extracting information from the Web. As for most of these systems [22], we base on the observation that data published in the pages of large sites usually come from a back-end database and are embedded within a shared HTML template. Therefore the extraction process consists of inferring a description of the shared template. Though this approach is applicable on Web documents as well, it does not exploit the hypertext structure of Web documents. Our work focuses on discovering this structure. 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 [6]. Additionally, information blocks always contain many frequent html elements, and also have a higher Degree of coherence (Doc) value than other blocks. That it to say, in Web pages there are many unique information features, which can be used to help extract information blocks. Therefore, it is possible to automatically extract information blocks by these unique features.

To this aim we define a DRE process as shown in Figure 1. In more details it is composed by the following steps

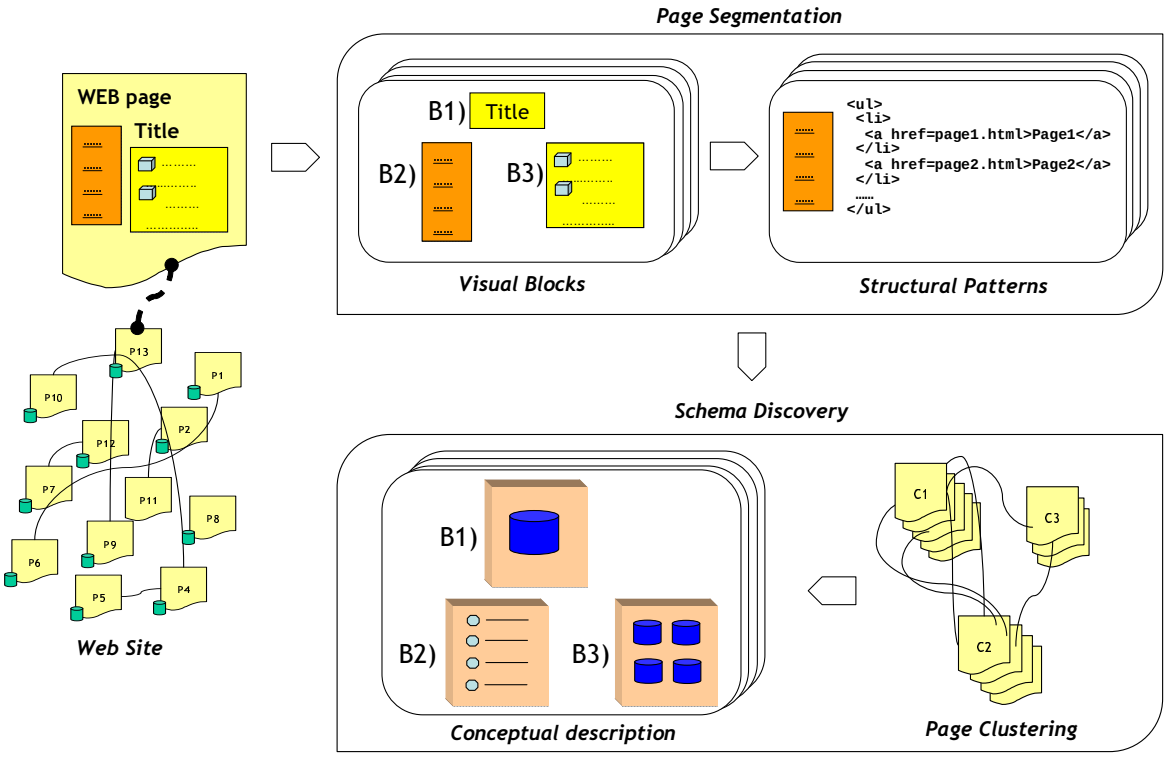

Figure 1: An overview of our DRE process

- *Page Segmentation*: several Web pages (representing a Web site) are segmented, each one, in several blocks respect with the visual perception of a user. Each resulting visual block of a Web page is isolated and, through a DOM analysis of the blocks, a set of structural patterns are discovered. Differently to other approaches based on exclusive implementations of DOM analysis solutions, this step combines a computer vision approach (to understand the perception of users) and a DOM structure extraction technique (which conveys the intention of Web authors).

- *Schema Discovery*: the resulting patterns, grouped in the visual blocks of each Web page, support the clustering of pages in the Web site. Each cluster represents aggregations of data semantically related and is represented by a set of structural patterns. Commonly it is generated a Wrapper to automate the extraction of patterns and to structure the Web content data related to them by an implementation into a storage model (i.e. relational model). Differently to these solutions, this step provides a conceptual model to represent Web content data at different level of details: content, navigation and presentation levels. The patterns of each cluster are mapped into constructs of our model. Finally, based on this conceptual representation, it is extracted a logical schema of the Web site.

In this section we will describe in detail the Page segmentation step, providing algorithms to compute the structural patterns. In the next section we will illustrate the schema discovery.

3.2 Page segmentation

We start exploiting the semi-structured nature of Web documents through document object model (DOM) trees extracted from the markups. Let's consider that DOM poorly reflects the

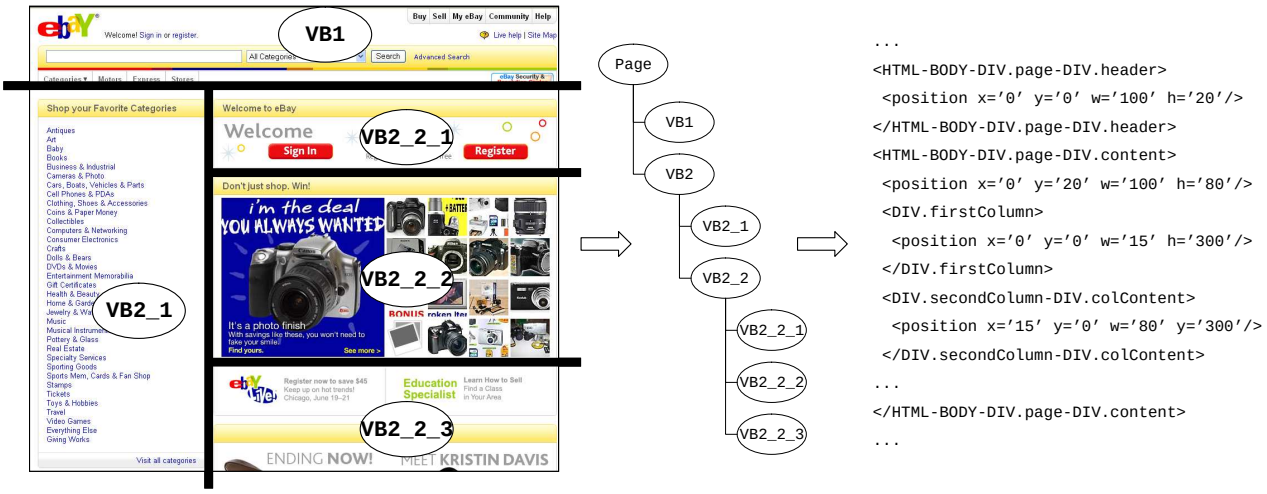

Figure 2: Visual Partitioning of a Web page

actual semantic structure of a page. However the visual page layout structuring is more faithful to reach semantic partitioning of a page. Therefore we make use of VIPS (Vision-based Page Segmentation) approach [8], basing on DOM tree and visual cues. Main idea is that semantically related contents are often grouped together and the page is divided into different information by using visual separators (e.g. images, lines, font sizes and so on). VIPS exploits the DOM structure and visual cues and extracts information blocks on its visual perceptions. The output of VIPS is a partitioning tree structure of *visual blocks* (VBs), for each Web page. The resulting VBs present a high Degree of coherence (Doc), meaning that they are homogeneous in the page. Then we assign an XML description to the tree: each VB is identified by the path in the DOM to reach it from the root of the page (considering also the available styling information of `class` or `id` referring to the associated Cascading Style Sheet or CSS) and characterized by the position in the page. For instance Figure 2 shows the resulting partitioning tree and XML description to the homepage of `www.ebay.com`.
Let's consider the visual block VB2_1 of Figure 2. It organizes information using an unordered list of items. Figure 3 shows an extract of DOM and CSS properties for VB2_1. Next phase is to analyze each identified VB and to discover 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 hash coding in a preorder traversal. The traversal generates a sequence V representing a vector of hash codes, as shown in Figure 3. 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 [26]. Algorithm 1 illustrates the pseudo-code of path-mark.

It manages a queue Q and the sequence V and returns a map M whereas 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 set a $windows$ (win) as the length of subsequences to analyze (line 7). It 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 emerging 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 foot. 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.

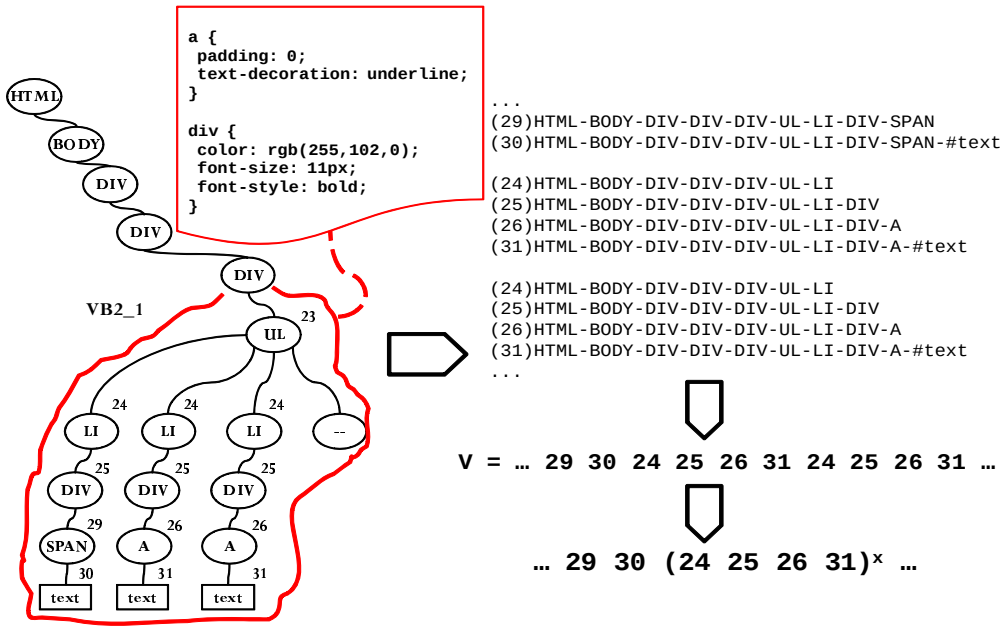

Figure 3: Pattern Searching

Algorithm 1 Path-Mark

Input: A visual block *VB*
Output: A Map of occurring patterns in *VB*, each one related to its occurences
begin
$V \leftarrow HASHPREORDER(VB)$ // *V is a sequence*
$EMPTY(M), EMPTY(Q)$ // *M is a map and Q a queue*
$seq\_length \leftarrow LENGTH(V)$
for win from 1 to $\frac{seq\_length}{2}$ **do**
    for $index$ from 0 to $seq\_length - win$ **do**
        $previous \leftarrow DEQUEUE(Q)$
        $actual \leftarrow SUBSEQUENCE(V, index, index + win)$
        $ENQUEUE(Q, actual)$
        if $actual = previous$ **then**
            $counter \leftarrow counter + 1$
            $internal \leftarrow index + win$
            while $internal < seq\_length$ **do**
                $next \leftarrow SUBSEQUENCE(V, internal, internal + win)$
                if $actual = next$ **then** $counter \leftarrow counter + 1$
                else $INSERT(M, actual, counter)$
                end if
                $internal \leftarrow internal + win$
            end while
        end if
    end for
end for
return M
end

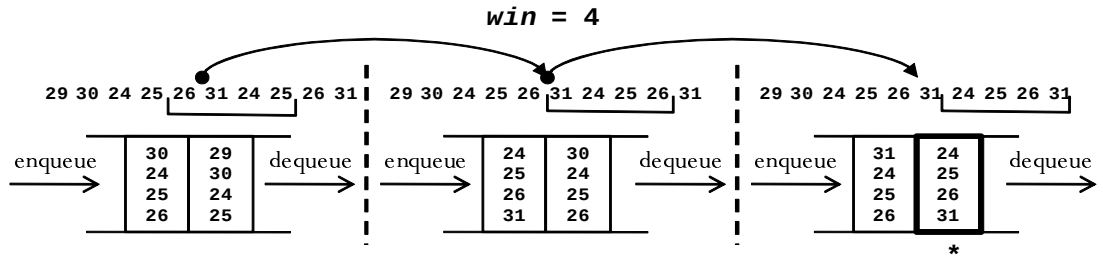

Figure 4: An execution of Path-Mark

Referring to the example of Figure 3, we show an execution of the algorithm in Figure 4 with $win = 4$. At the end, our algorithm produces a resulting grouping 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. Also in this case we produce an XML description containing information of patterns occurring in each VB as follows.

```
... <HTML-BODY-DIV.page-DIV.content-DIV.firstcolumn>
 <position x="0" y="0" w="15" h="300"/>
 <Element id="1" containsRepeatedPattern="true">
  <Item class="menuItem" content="Antiques"
                         item="UL-LI-DIV-A-"/>
  <Item class="menuItem" content="Art"
                         item="UL-LI-DIV-A-"/>
  <Item class="menuItem" content="Baby"
                         item="UL-LI-DIV-A-"/>
  <Item class="menuItem" content="Books"
                         item="UL-LI-DIV-A-"/>
  ...
 </Element>
</HTML-BODY-DIV.page-DIV.content-DIV.firstcolumn> ...
```

4 Schema Discovery

4.1 A Web Site Model

To abstract the main structural properties of an HTML page, we provide a Web Site Model (WSM). It represents a common abstract representation (or metamodel) of the various modeling approaches to describe a Web application (for a survey see, e.g. [23]). In particular we can individuate principal constructs of each model, used to describe information in a Web page, and classify them, defining a set of basic constructs.

According to the classical organization of a Web resource (content, navigation, and presentation) [9], our metamodel provides a hypertext definition describing a way to organize the elements of the content into a hypertext. Figure 5 presents a simplified class diagram of WSM. More precisely, it organizes a page in a set of *meta containers* related by *links*. A metacontainer is *atomic* if it represents an atomic portion of a linked page and includes a direct reference to

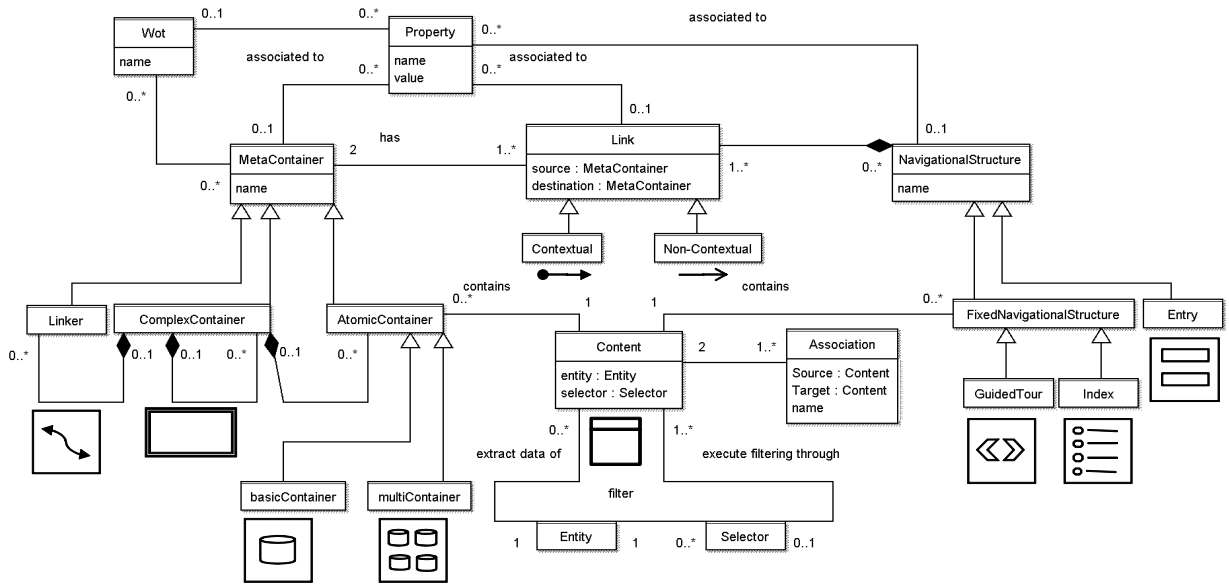

Figure 5: Class diagram of our Web Site Model

the elements of a content from which it takes data. It can be *basic* if it shows information about a single object (e.g. an instance of an entity) or *multi* if it shows information about a set of objects. A metacontainer is a *linker* if it contains an anchor between containers. Otherwise a metacontainer is *complex* if it is articulated in other containers (atomic, complex or linker). So a page represents a complex container. Each metacontainer is identified by a *name* and presents different *properties* (e.g. the attributes of the content to include). We can surf the containers through several *navigational structures* such as *Indexes* to show a list of objects without presenting the detailed information of each one, *Guided Tours* to show commands for accessing the elements of an ordered set of objects, and *Entries* to show edit field for inputting values used for searching within a set of objects meeting a condition. Metacontainers do not exist in isolation. They have to be connected by links to form a hypertext structure. Links can be *contextual*, if they connect containers carrying some information from the source container to the destination one (i.e. they have assigned properties in the model). Otherwise they are non-contextual if they connect containers in a totally free way. Finally, a *logical style sheet* describes the actual look and feel of metacontainers mentioned in a hypertext definition. It is based on a predefined set of *Web object types* (Wots). Possible Wots are text, image, video, form, and so on. Each Wot is associated with a set of presentation attributes (i.e. properties): they identify possible styles (e.g. font, color, spacing, position) that can be specified for it. Each attribute is associated with a domain of possible values for the attribute. Figure 5 illustrates the symbolic notation of each concept.

For instance, Figure 6 shows an example of hypertext definition. It organizes items of an e-commerce Web site. The home page contains several discounted items to surf using an Index, and different item categories to look through a Guided Tour in a separate page.

4.2 Matching Web entity blocks

In the previous section we presented a technique to segment a Web page into structural blocks representing aggregations of data semantically related. The final step is to discover a logical

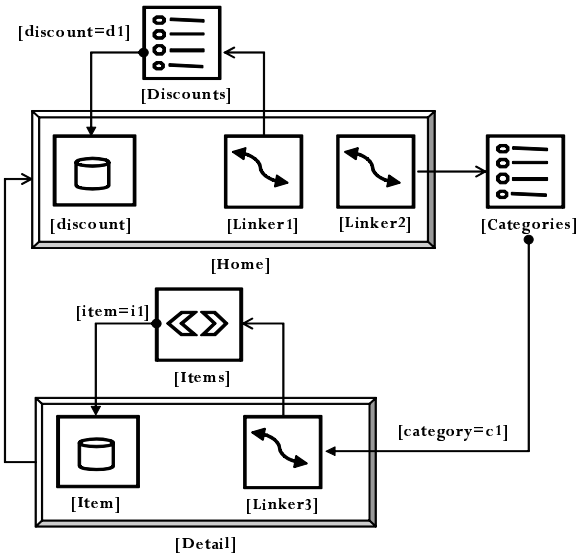

Figure 6: An example of hypertext definition

schema matching the discovered patterns with the constructs of the Web Site Model described above. The entire process is shown in Figure 7.

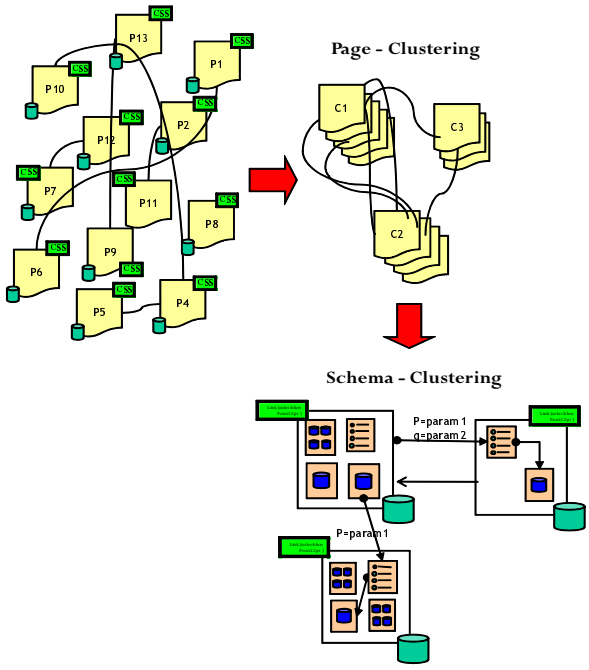

Figure 7: Schema discovery

For our purposes, 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 a collection of patterns $pt_1, pt_2, \ldots$, identified by the path-mark algorithm, shown in the previous section. So we define *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 patterns node-to-link together with all the URLs that share that pattern. For instance consider the Web pages *Page1*, *Page2* and *Page3* at left side of Figure 8. They are described by the hierarchical tree

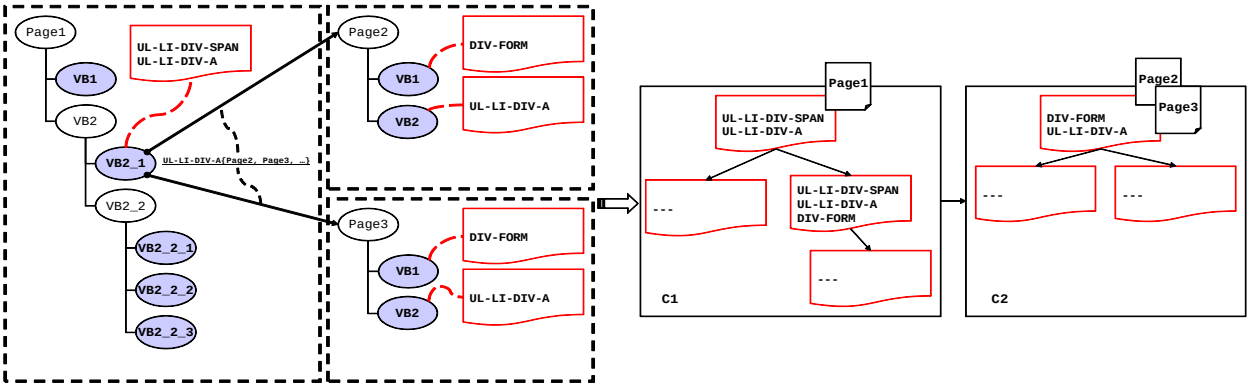

Figure 8: From Pages to Clusters

of visual blocks, and each block has associated the set of patterns, individuated by the Algorithm 1. *Page2* (*Page3*) presents a page schema as the set {`DIV-FORM, UL-LI-DIV-A`}, and the link collection {`UL-LI-DIV-A`$\{url_1, url_2, \ldots\}$}. *Page1* presents the link collection {`UL-LI-DIV-A`$\{Page2, Page3, \ldots\}$}

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 for each pattern pt in ps_1, it exists a pattern pt in ps_2.

The distance between two page schemas is defined as the normalized cardinality of the symmetric patterns set difference between the two schemas. Let's 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 $ps_1 = ps_2$ (identical schemas), 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. More in detail, 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 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 [1]. For instance at right side of Figure 8 there are the clustering of the three pages *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 to schema discovery is to compute 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.

[1] We have experimentally determined $dt = 0,4$

We have designed an algorithm that builds a set of clusters incrementally. Algorithm 2 shows the pseudo code. The input of the algorithm is a starting Web page p_0 (i.e. the home page), which represents the first member of the first cluster in the set CL (line 5). The output is the set of computed clusters CL.

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**

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 to by its links is fetched (line 9)[2]. 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 respect to the threshold dt. We remark the main idea to maintain in a cluster C a set of pages having page schemas with a distance from the root r_C lower than dt. Intuitively we extract the page schema ps of p, by using the path-mark algorithm, and select the cluster C in CL having a root with the minimum distance from ps (lower than dt). If there is no cluster satisfying these properties then 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.

- if there is no child n of the root r_C of C such that $n \lhd ps$, then (i) ps becomes the child of r_C, and (ii) each child n of r_C such that $p \lhd n$ becomes child of ps;

[2]We assume that is sufficient to follow a subset of potentially large set of links to determine the homogeneity properties of the entire collection

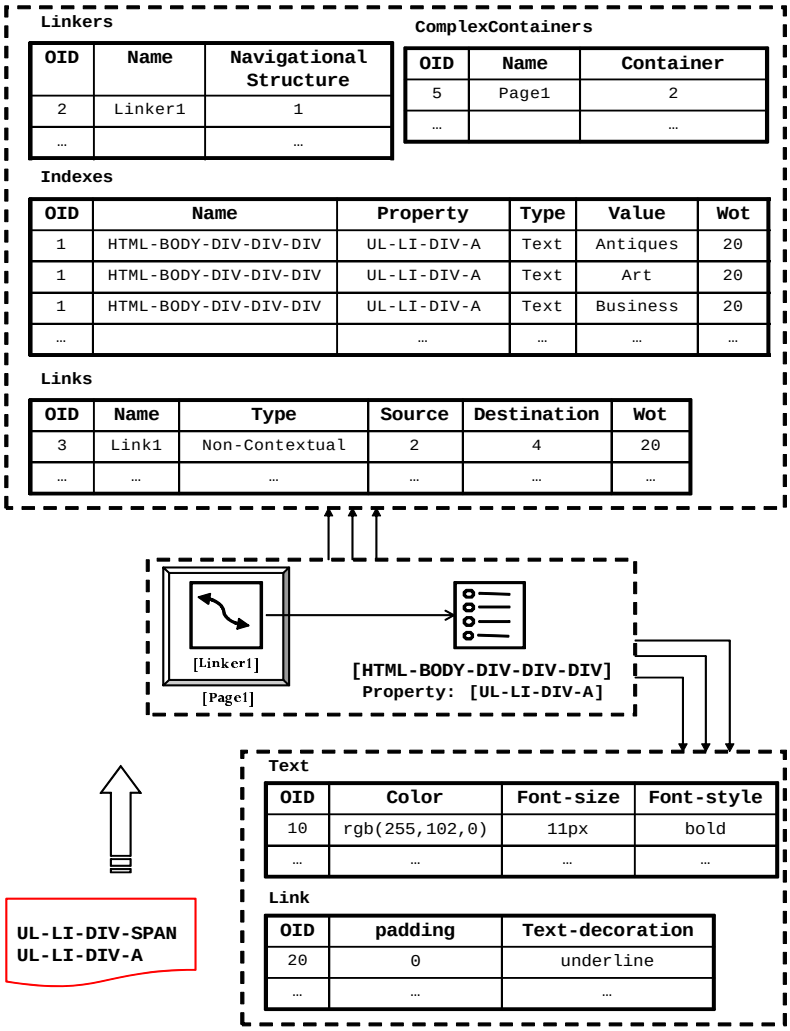

Figure 9: An example of Web entity blocks

- otherwise, we insert ps in the sub-tree of C having as root the child n of r_C such that (i) $n \lhd ps$, and (ii) 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 through heuristics by visiting the fewest possible pages. We assign higher priority to link collections that have many instances relative to the total number of outgoing links for a cluster. This means that long lists in a page are likely to point pages with similar content (i.e. they are generated by a program), and therefore the next set of pages will provide high support to their common cluster.

The final step of schema discovery is associating a representation of our Web Site Model to each cluster. The idea is to indicate 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

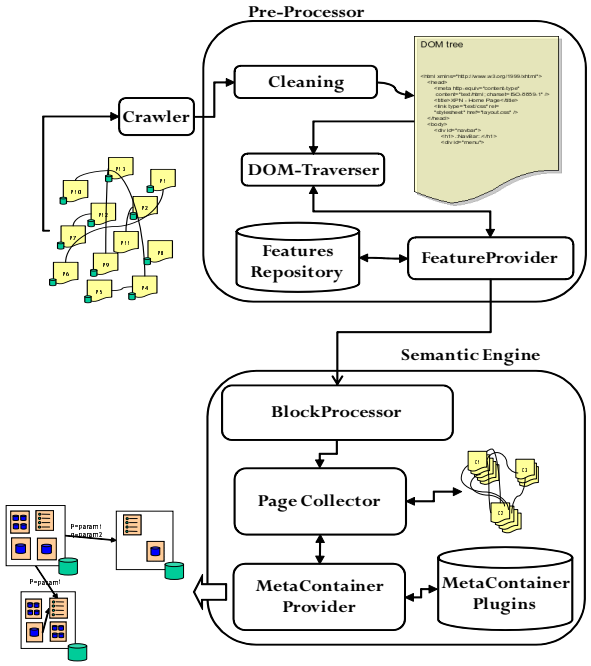

Figure 10: An Architecture of reference

into a metacontainer using a particular navigational structure (Index, Guided Tour or Entry). We fix a set of heuristics for each construct of our metamodel. Referring to the example of Figure 3 we can map the pattern `UL-LI-DIV-A` into an Index, whereas we have a heuristic that maps a pattern `UL-LI-#-A` with an Index. Each metacontainer (navigational structure) 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 respect to the information content associated to each pattern and the presentation respect 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. For instance Figure 9 shows the Web entity blocks associated to the root of cluster C_1 shown in Figure 8 and the relative organization (i.e. implementation) of metadata in a relation DBMS.

5 Experimental Results

On the basis of theoretical results and practical techniques above described, we have designed a tool for data reverse engineering of data intensive Web applications, called `ReverseWeb`. Figure 10 shows an architecture of reference.

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 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

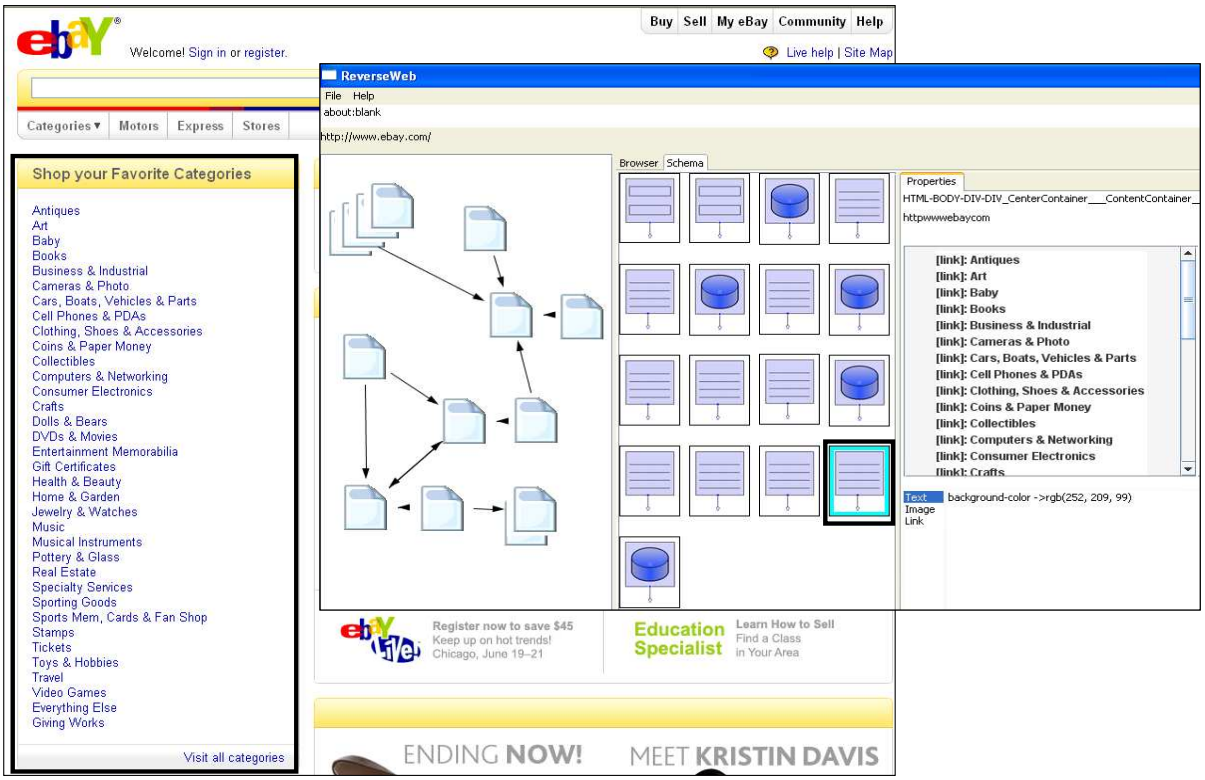

Figure 11: ReverseWeb at Work

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 was developed multithreading through an internal browser. The GUI was realized by using SWT toolkit (`http://www.eclipse.org/swt/`), designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented, and NetBeans Visual Graph Library (`http://graph.netbeans.org/`). All algorithms and heuristics have been implemented in Java. Finally we use CSS steadystate Library (`http://cssparser.sourceforge.net/`) to parse the presentation properties of a page. Figure 11 shows a snapshot of the tool at work on the Web site `www.ebay.com`. An alpha version of the tool will be available soon at `http://mais.dia.uniroma3.it/ReverseWeb`.

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. *http://web.dia.uniroma3.it/* (DIA) and *http://www.wordrefe- rence.com* (WORD) that are the Department of Informatics and Automation of Roma Tre University and the Dictionary translator Web Site respectively;

2. *http://www.buy.com* (BUY) and *http://www.ebay.com* (EBAY) that are two famous E-commerce Web sites;

3. *http://www.nba.com* (NBA) that is a famous Sport Web site.

	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

Table 1: Experimental Results on 1000 pages

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 entity 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 entity blocks as follows. First of all consider 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. Thus 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.

Therefore, 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 of nodes for page. This means the percentage of DOM nodes in a page that were involved in the final schema. This coefficient is in a range [0,1]. If DRE quality is too closed to zero, this means that the system wasn not able to process and assign a semantic to the blocks, otherwise if DRE quality is too closed to one, the system had difficulties to prune unmeaningful blocks. 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 conseguence 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: Figures 12 and 13 illustrate the number of Web entity 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 12 is shown the trend of the number of Web entity 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

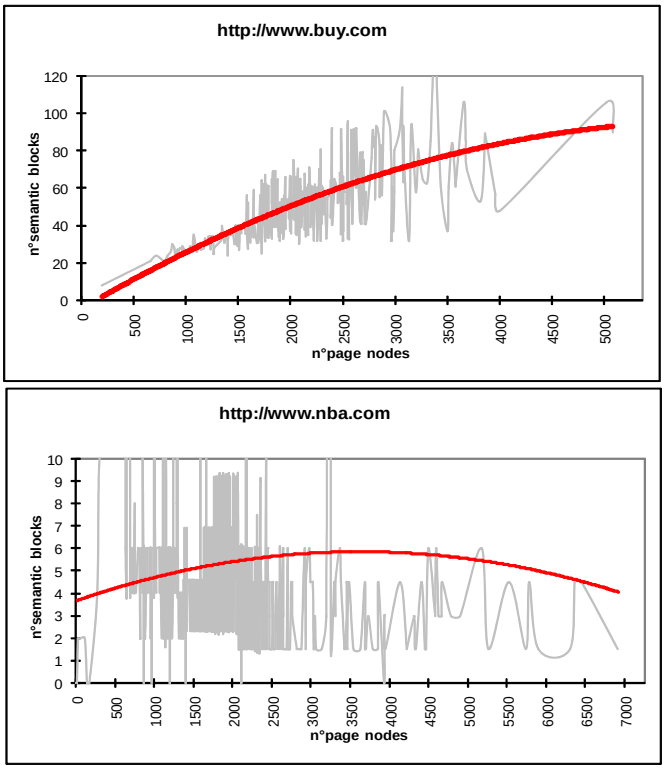

Figure 12: Number of Web entity blocks

is due to common structure of the published information (basketball teams features) and this is close to the reality.

This regularity is also supported by the stable average of the elapsed time, shown in Figure 13. Conversely, BUY presents a variable structure of pages with an increasing trend of Web entity blocks and elapsed times. This is due to the different structure of published information with (i.e. 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 14 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 5,1.

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, concentrating on the data aspect of the system that is the organization. We presented a collection of methods and tools as support to determine the structure, function, and meaning of data in a Web site. We used *structured* techniques as page segmentation and model building techniques involving model construction and analysis of the existing situations. Finally we evaluated the performance of our framework by implementing a tool, called ReverseWeb, and facing several experiments

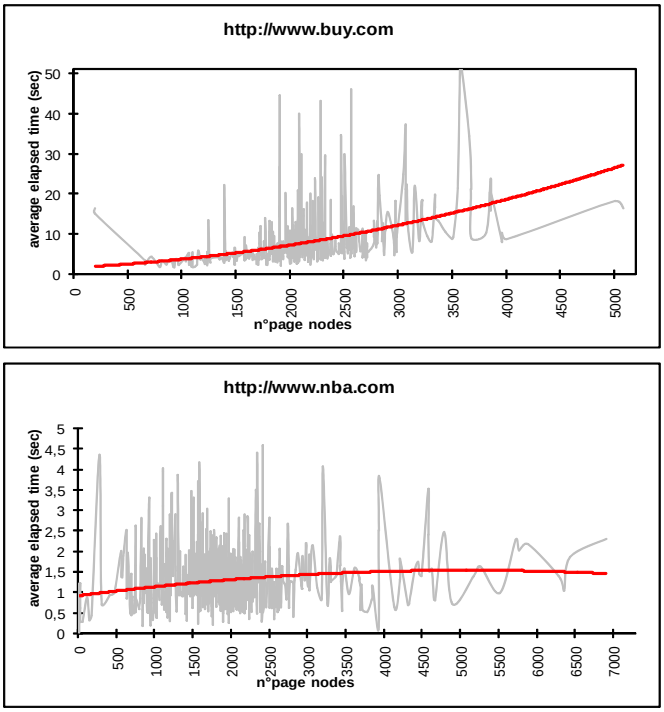

Figure 13: Average elapsed time

on different Web sites. There are several interesting future directions. We are improving the segmentation step, introducing new features in the preprocessing phase. We want to introduce a notion of polymorphism to optimize the mapping of patterns with Web entity blocks. To this aim, we can define a common grammar based heuristic. Finally we are refining the clustering technique, defining a more sophisticated algorithm based on the notion of distance between pages and exploiting the information at the segmentation phase.

References

[1] P. H. Aiken. Reverse Engineering of Data. In *IBM Systems Journal, 37(2)*, 1998.

[2] G. Antoniol, G. Canfora, G.Casazza, and A. De Lucia. Web Site Reengineering using RMM. In *Proc. of International Workshop on Web Site Evolution, Zurich, Switzerland*, 2000.

[3] R. Baumgartner, S. Flesca, G. Gottlob. Visual Web Information Extraction with Lixto. In *Proc. of the 27th International Conference on Very Large Data Bases (VLDB'07), Roma, Italy*, 2001.

[4] P. Benedusi, A. Cimitile, U. de Carlini. Reverse engineering processes, design document production, and structure charts. In *Journal of Systems and Software 19(3): 225-245*, 1992.

[5] S. M. Benslimane, D. Benslimane, M. Malki, Y. Amghar, H. S. Hassane. Acquiring owl ontologies from data-intensive web sites. In *Proc. of International Conference on Web Engineering (ICWE'06), Palo Alto, California, USA*, 2006.

[6] M. L. Bernard. Criteria for optimal web design. *Document available at http://psychology.wichita.edu/optimalweb/text.htm*, 2006.

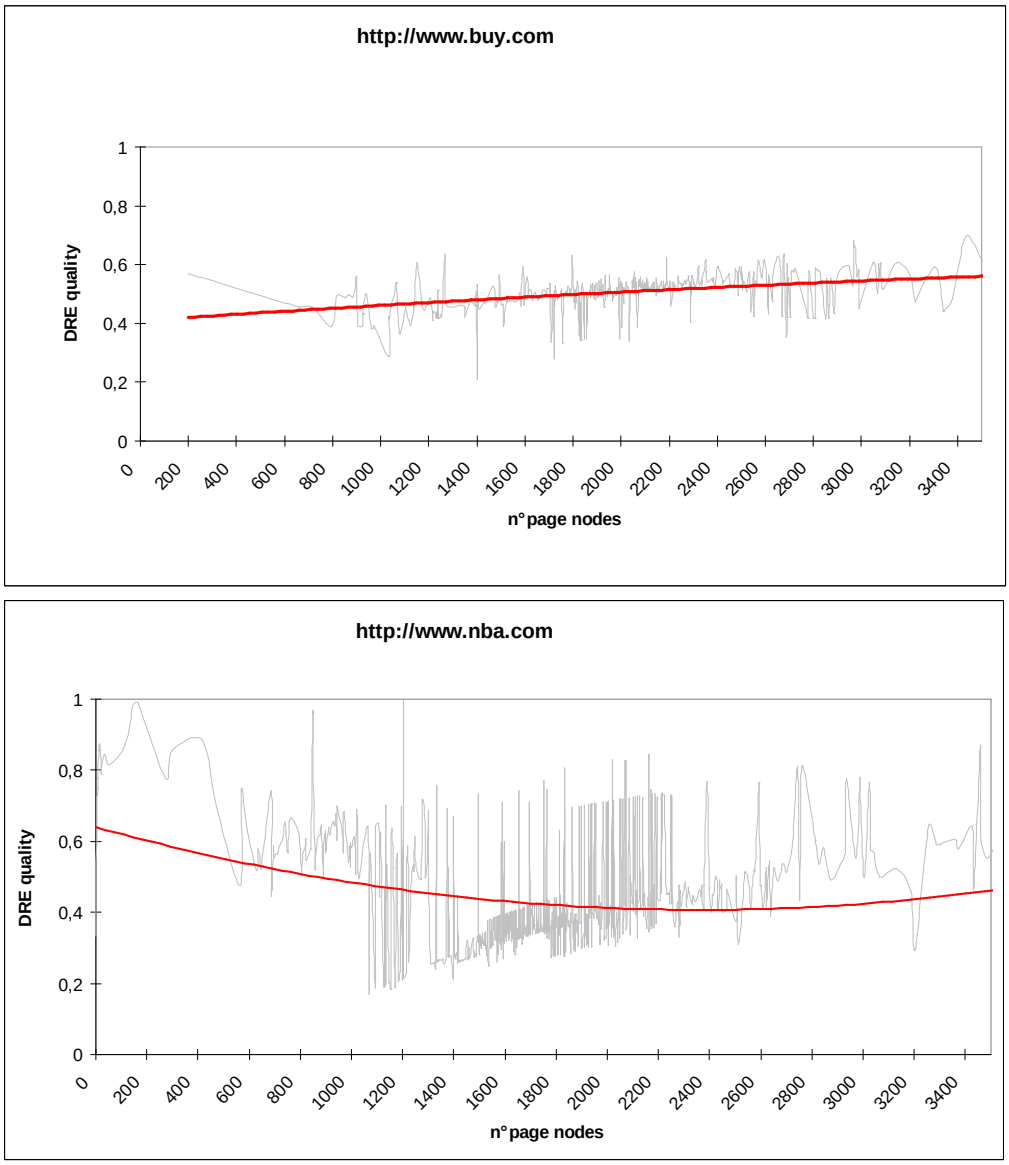

Figure 14: DRE quality

[7] D. Bouchiha, M. Malki, and S.M. Benslimane. Ontology based Web Application Reverse Engineering Approach. In *INFOCOMP Journal of Computer Science, 6(1): 37-46*, 2007.

[8] D. Cai, S. Yu, J. R. Wen and W. Y. Ma. Extracting Content Structure for Web Pages based on Visual Representation. In *Proc. of the 5th Asian Pacific Web Conference (APWeb'03), Xian, China*, 2003.

[9] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Matera. Designing Data-Intensive Web Applications. *Morgan-Kaufmann*, 2003.

[10] S. Chakrabarti. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In *Proc. of the 10th International World Wide Web Conference (WWW'01), Hong Kong, China*, 2001.

[11] S. Chakrabarti, M. Joshi, V. Tawde. Enhanced Topic Distillation Using Text, Markup Tags, and Hyperlinks. In *Proc. of the the 24th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'01), New Orleans, Louisiana*, 2001.

[12] E. J. Chikofsky. The Necessity of Data Reverse Engineering. *Foreword for Peter Aikens Data Reverse Engineering, pages 811. McGraw Hill*, 1996.

[13] E. J. Chikofsky, J. H. Cross. Reverse Engineering and Design Recovery: A Taxonomy. In *IEEE Software 7(1): 13-17*, 1990.

[14] S. Chung, Y. S. Lee. Reverse Software Engineering with UML for Web Site Maintenance. In *Proc. of the 1th International Conference on Web Information Systems Engineering (WISE'00), Hong Kong, China*, 2000.

[15] J. Conallen. Building Web Applications with UML. *Addison Wesley*, 1999.

[16] G. A. Di Lucca, A. R. Fasolino. Testing Web-based applications: The state of the art and future trends. In *Information & Software Technology 48(12): 1172-1186*, 2006.

[17] G. A. Di Lucca, A. R. Fasolino, F. Pace, P. Tramontana, U. de Carlini. WARE: A Tool for the Reverse Engineering of Web Applications. In *Proc. of the 6th European Conference on Software Maintenance and Reengineering (CSMR'02), Budapest, Hungary*, 2002.

[18] G. A. Di Lucca, A. R. Fasolino, P. Tramontana. Reverse engineering Web applications: the WARE approach. In *Journal of Software Maintenance 16(1-2): 71-101*, 2004.

[19] B. Du Bois. Towards a Reverse Engineering Ontology. In *Proc. of the 2th International Workshop on Empirical Studies in Reverse Engineering (WESRE'06), Benevento, Italy*, 2006.

[20] J.L. Hainaut, J. Henrard, J.M. Hick, D. Roland, and V. Englebert. The nature of data reverse engineering. In *Proc. of Data Reverse Engineering Workshop (DRE'00), Zurich, Switzerland*, 2000.

[21] J. Kesselman, J. Davidson, B. Chang, M. Champion, A. Le Hors, A. Diaz, A. Heninger, R. Whitmer, P. Le Hgaret, T. Pixley, J. Sorensen and La. Wood. Document Object Model (DOM) Requirements. *W3C Note*, 2004.

[22] A. Laender, B. Ribeiro-Neto, A. Da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. In *ACM SIGMOD Record, 31(2): 84-93,* 2002.

[23] W. Retschitzegger, and W. Schwinger. Towards Modeling of DataWeb Applications - A Requirements Perspective. In *Proc. of the Americas Conference on Information Systems (AMCIS'00), Long Beach California*, 2000.

[24] F. Ricca and P. Tonella. Understanding and Restructuring Web Sites with ReWeb. In *IEEE Multimedia 8(2): 40-51*, 2001.

[25] A. Sahuguet, F. Azavant. Building Intelligent Web Applications Using Lightweight Wrappers. In *Data Knowledge Engineering Journal 36(3):283-316*, 2000.

[26] T. Tao, A. Mukherjee. LZW Based Compressed Pattern Matching. In *Proc. of the 14th Data Compression Conference (DCC 2004), Snowbird, UT, USA*, 2004.

[27] J. Vanderdonckt, L. Bouillon and N. Souchon. Flexible reverse engineering of Web Pages with VAQUISTA. In *Proc. of the 8th Working Conference on Reverse Engineering (WCRE'01), Suttgart, Germany*, 2001.