A TOOLKIT TO RESTRUCTURE STATIC WEB SITES INTO DYNAMIC WEB APPLICATIONS

Paolo Tonella and Filippo Ricca
ITC-irst, Centro per la Ricerca Scientifica e Tecnologica,
38050 Povo (Trento) – ITALY
{tonella, ricca}@itc.it

Abstract

This chapter presents a Toolkit that has been designed to support a restructuring process aiding the migration of static Web sites into dynamic Web applications. In particular, this Toolkit replaces a set of static Web pages having a similar structure with a server script that generates dynamic pages at run time. The dynamically generated pages are equivalent to those in the purely static version of the Web site. This chapter describes the main technical features of the toolkit and its main functionalities. Moreover, the advantages of the tool-generated dynamic version of the site are discussed together with the manual interventions needed to produce it. A real-world case study describing how the toolkit can be actually used in the migration from a static Web site to a dynamic Web application is reported.

1. Introduction

The first generation of Web sites consisted typically of a collection of static HTML Web pages containing only fixed information to be shown to the users. Typical examples are Web sites advertising company products (e.g., providing a list of product-cards) and giving information about the company.

Such Web sites have been progressively replaced by e-commerce Web applications that are more dynamic, with functionalities devoted to accessing the data interactively, performing transactions on-line, and exchanging information. Among the others, one of the advantages of dynamic Web sites is that content (data) and presentation can be separated. This has remarkably positive effects on the Web site maintainability and overall quality. In fact, if data are stored separately (e.g., in a database), and Web pages are created dynamically, based on a fixed template, the update of information can be directly performed on the separate data storage. Programs for the dynamic generation of the pages may also need to be modified, but still with the advantage that formatting is not intermixed with content. Moreover, using a single dynamic program to generate a whole set of pages ensures a consistent structure and formatting of the result.

For multilingual Web sites (i.e., sites where the information is supplied in more than one language, e.g. Italian, English, French etc.), the advantages of the dynamic approach are even more evident, since these sites have the additional problem of the consistency of structure and content across languages (Tonella *et* al., 2002). The dynamic generation of the pages in all languages from a common template, as well as the separation of the multilingual content from the page structure, alleviate the consistency problem and simplify Web site evolution.

Several existing Web sites have not yet moved to the next generation of dynamic sites. There are several examples of sites in which the products commercialized by a company are presented in static Web pages, where a given HTML structure is replicated for each product-card, with product-specific values in each page. A consequence is that typically inconsistencies are present. Different products are described in a different way, with a different formatting and page structure. Even inconsistent data might exist, especially if the site is multilingual. For these companies a tool-supported process to migrate (multilingual) product-cards into a database could be useful.

In this book chapter we propose a Toolkit aimed at identifying and migrating static Web pages, for example product cards, that can be transformed into dynamic ones. At the end of this process, a script generates the pages of the Web site dynamically, by accessing a database containing the variable data in the migrated pages.

The chapter is organized as follows: Section 2 describes the restructuring Toolkit. In particular, this section illustrates the architecture and the functionalities that our Toolkit implements. The restructuring process is first presented in the simpler case of a mono-lingual Web site, and then it is extended to the general case of a multi-lingual site, in Section 3. A case Study is provided in Section 4, where the usefulness of our toolkit is discussed. Related works are commented in Section 5 while the last section is devoted to conclusions.

2. The Restructuring Toolkit

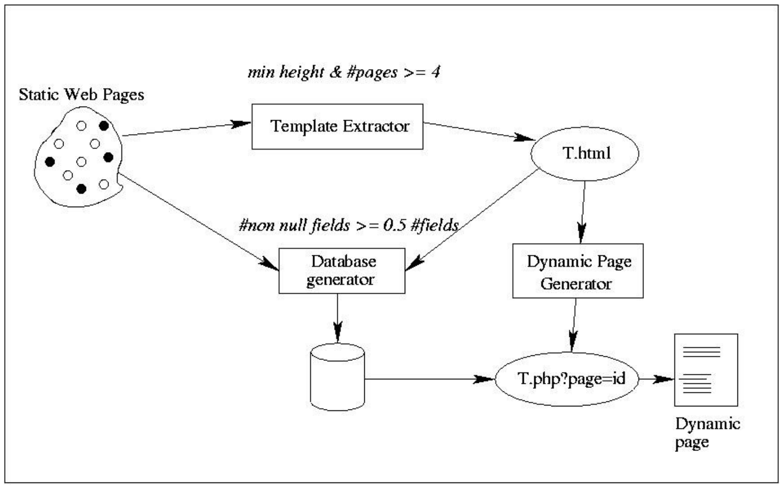


Figure 1. The Restructuring Toolkit

The Toolkit depicted in Fig. 1 replaces a set of static Web pages having the same structure with a unique server script (*T.php*) that generates dynamic pages. The dynamic pages generated at run time have a fixed part (the template *T.html*) and a variable part that is built using the information migrated into the database.

The *Template Extractor tool* exploits clustering to recognize a common structure of the Web pages. It provides the candidate template to be used in the dynamic generation of the pages.

The *Database Generator too*l compares the original pages with the template *T.html* providing the records (variable part) to be inserted into the database.

The output of the *Dynamic Page Generator tool* is a script that generates the migrated pages dynamically, from the template and the database.

Manual intervention is limited to the refinement of the constructed template and database. The toolkit has been implemented using the Java programming language and TXL (Cordy *et* al., 2002).

2.1 Template Extractor

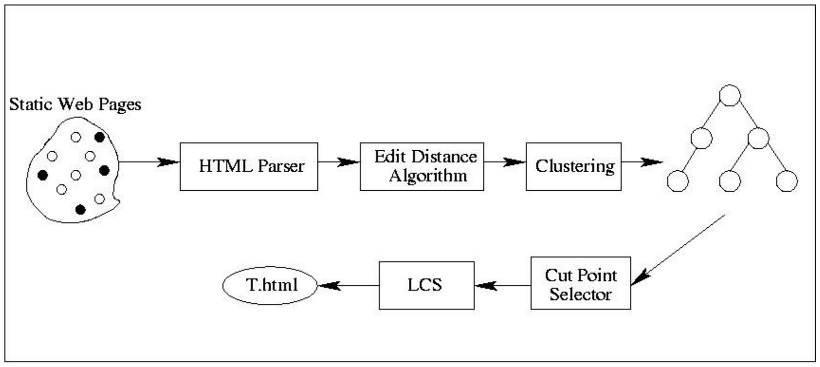


Figure 2. The Template Extractor Tool

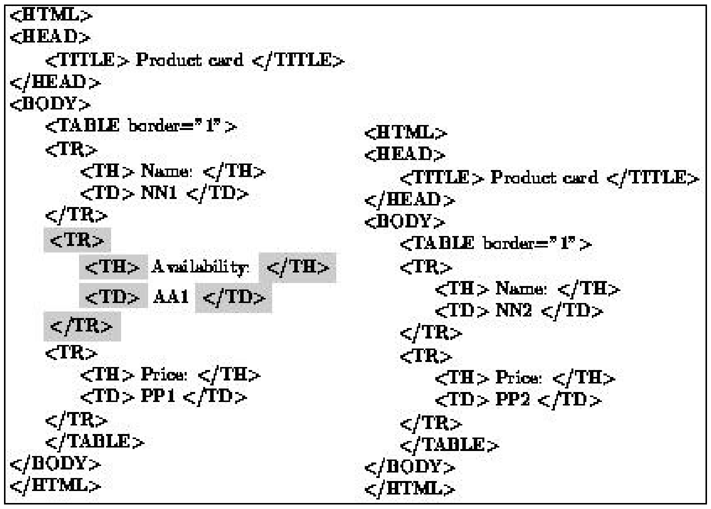
We expect that the static Web pages to be restructured into a database be similar (but not necessarily identical) from a structural point of view. The typical example is a set of product-cards, where only product-specific information varies.

Clustering is used to recognize Web pages with a common structure. A natural choice of clustering technique is the sibling link approach (Anquetil and Lethbridge, 1999) based on a structural measure of similarity between Web pages.

The *structural edit distance (*Atallah, 1999*)* among the Web pages is a good choice to measure the dissimilarity of pages (its complement gives the required similarity measure). For example, product-cards have typically a low structural edit distance (high similarity).

An HTML parser, embedded into the *Template Extractor tool,* produces the abstract syntax trees of the Web site pages analyzed. The JavaCC HTML parser written by Brian Goetz (www.quiotix.com) has been employed for this work.

In order to compare the syntax trees associated with two pages, a tree visit is performed, during which a list representation of the tree is generated. The edit distance of two pages P and Q is defined as the number of edit operations that transform the list representation of a page P into the list associated with the page Q. The set of allowed edit operations on sequences consists of element deletion, insertion and update (the latter weighted 2, since it is equivalent to a deletion followed by an insertion).



```
<HTML>
<HEAD>
    <TITLE> Product card </TITLE>
</HEAD>
<BODY>
    <TABLE border="1">
    <TR>
        <TH> Name: </TH>
        <TD> NN1 </TD>
    </TR>
    <TR>
        <TH> Availability: </TH>
        <TD> AA1 </TD>
    </TR>
    <TR>
        <TH> Price: </TH>
        <TD> PP1 </TD>
    </TR>
    </TABLE>
</BODY>
</HTML>
```

```
<HTML>
<HEAD>
    <TITLE> Product card </TITLE>
</HEAD>
<BODY>
    <TABLE border="1">
    <TR>
        <TH> Name: </TH>
        <TD> NN2 </TD>
    </TR>
    <TR>
        <TH> Price: </TH>
        <TD> PP2 </TD>
    </TR>
    </TABLE>
</BODY>
</HTML>
```

Figure 3. Pages p.html and q.html

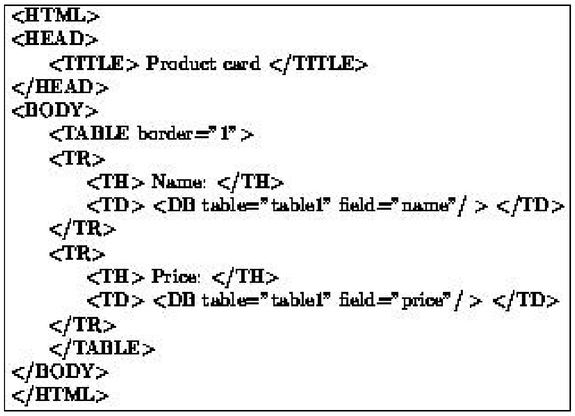
In the example in Fig.3, it is easy to see that the tag sequence associated with *p.html* can be transformed into that associated with *q.html* by deleting the six elements with gray background: TR, TH, /TH, TD, /TD, and /TR. Therefore the two pages have a structural edit distance of 6.

In the literature there exist several different clustering algorithms (Wiggerts, 1997), with different properties. We have implemented the agglomerative hierarchical clustering algorithm, where the output is a hierarchy of possible clusterings.

The next step of the Template Extractor is the selection of a cluster from which a template page with all fixed information and common HTML tags can be extracted.

The algorithm (*Cut point selector*) that has been defined to automate such a choice takes into consideration cut points of the hierarchy at increasing height (at height 0 all pages are in singleton clusters, at the top all pages are in one cluster). When a cluster is encountered which contains a number of pages greater than or equal to a given threshold (set to 4 for the experiments), such a cluster is selected for the computation of the template.

At this point the Longest Common Subsequence (*LCS*) algorithm is used. LCS is applied to the set of pages of the selected cluster and the result is the sequence of tags and texts that are common to all pages, i.e., the template (*T.html*) of the dynamic pages to be generated. Note that while for clustering the text between tags is ignored, computation of the LCS involves the comparison of the text between tags in addition to the tags. When a difference is detected, the template includes a placeholder for a value retrieved from the database, consisting of a new tag DB with the attributes *table* and *field* for the identification of the database entry to retrieve. A page identifier is used to select a given record from the database table.



```
<HTML>
<HEAD>
    <TITLE> Product card </TITLE>
</HEAD>
<BODY>
    <TABLE border="1" >
    <TR>
        <TH> Name: </TH>
        <TD> <DB table="table1" field="name"/ > </TD>
    </TR>
    <TR>
        <TH> Price: </TH>
        <TD> <DB table="table1" field="price"/ > </TD>
    </TR>
    </TABLE>
</BODY>
</HTML>
```

Figure 4. Example of template obtained as the LCS from the pages in Fig. 3

Fig. 4 shows the LCS produced by comparing the pages *p.html* and *q.html* of Fig.3.

2.2 Database Generator

The Database Generator provides the records to be inserted into the database.

Record fields are initially approximated as the set of fixed text strings appearing in the template. However, not all of them are actually fields. Those that are not followed by a variable text (the field value) can be disregarded. Moreover, all fixed text strings for which only a minority of the records (less than 50%) has a non null value are disregarded as well.

For each record, the values of the fields to be inserted into the database are determined as the variable text in each page that appears after the fixed text recognized as a field name.

The TXL function that generates the couples field-value for each page follows.

```
function buildFieldPlusValue page [repeat element] field [htmlText]
        replace [couple]
              _
        deconstruct * page
              field restWithValue [repeat element]
        deconstruct * restWithValue
              value [htmlText] rest [repeat element]
        construct c [couple]
              field : value
        by
              c
end function
```

The two searching deconstructors[1] (Cordy, 2004) applied in sequence inside the TXL function *buildFieldPlusValue* respectively check for the presence of an element (of type *htmlText*, i.e. text inside HTML tags) equal to the parameter *field*, and recover the next following element of type *htmlText*, which is used as candidate value for the given field.

During the phase of Database generation, not only the pages in the selected cluster are migrated to the database: all Web site pages are re-examined, and each time the number of non-null field values that can be determined for a page is greater than or equal to half of the total number of fields, the static page is converted into a dynamic one, and the variable information it contains becomes a new record inserted into the database. For each page, a page identifier is used as primary key of the related database record.

Let us suppose to analyze a Web site with the following six pages: *p.html* and *q.html* as in Fig.3, *p'.html* equal to *p.html* but with different values (NN1', AA1', PP1' instead of NN1, AA1, PP1). *q'.html* and *q''.html* equal to *q.html* but with different values (respectively NN2', PP2' and NN2'', PP2'' instead of

[1] In TXL functions may contain deconstructors (keyword deconstruct). Deconstructors are used to break variables into smaller pieces using a more refined pattern. If a deconstructor pattern does not match, then the function is considered to have not matched its pattern. Searching deconstructors (keyword deconstruct *) are a specialization of deconstructors. They can be used to search for and take a subtree of the deconstructed tree. A searching deconstructors takes the following form:

```
deconstruct * [type] varName
    pattern
```

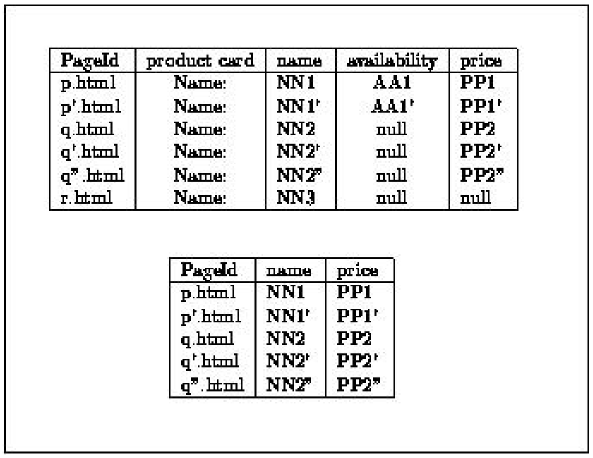
The semantics of this construct is: searching the tree bound to varName for the first subtree that matches the pattern and binding the pattern variables accordingly.

NN2, PP2). Finally *r.html* with a structure very similar to *q.html*, but without "Price:" and "PP2" inside the <TR> tag. If the cluster chosen for *Template extrac*tion consists of the pages *p.html* and *p'.html*, the generated template *T.html* is similar to the template of Fig.4 with an extra <TR> row (labeled "availability").

Given this template, the list of potential fields is composed of four elements: "product card", "name", "availability", and "price".

During the phase of *Database generation*, some of the candidate fields (columns) and one of the candidate records (rows) of the initial version of the database (Fig.5, top) are disregarded. Precisely column 1 ("product card") is disregarded because the related values are not variable (all of them are equal to "Name:"), and column 3 ("availability") because the number of non-null values is less than 50%. Row 6 (*r.html*) is disregarded for the same reason. Thus, page *r.html* will remain a static page.

The final result of database generation is shown at the bottom of Fig.5.



PageId	product card	name	availability	price
p.html	Name:	NN1	AA1	PP1
p'.html	Name:	NN1'	AA1'	PP1'
q.html	Name:	NN2	null	PP2
q'.html	Name:	NN2'	null	PP2'
q".html	Name:	NN2"	null	PP2"
r.html	Name:	NN3	null	null

PageId	name	price
p.html	NN1	PP1
p'.html	NN1'	PP1'
q.html	NN2	PP2
q'.html	NN2'	PP2'
q".html	NN2"	PP2"

Figure 5. An example of database generation before (top) and after (bottom) disregarding some records and fields

2.3 Dynamic Page Generator

Once template and database have been generated, all static pages migrated to the database can be replaced by a server side script that generates them at run time (*T.php* in Fig.1). Each original request of a migrated page will be replaced by an invocation of *T.php*, with a parameter *page=id* that identifies the information of interest. The server side script uses this parameter to select the record with primary key equal to *id* from the database table generated before. Each time a *DB* tag is encountered by *T.php*, the database is accessed to retrieve the record indexed by *id*. The field specified as a *DB* attribute is produced in output, as a replacement of the tag.

When more than one group of static pages can be migrated to dynamic pages (e.g., product cards for different product categories), the process in Fig.1 is iterated, until no useful cluster can be determined. In this way, all site portions that share a common structure are identified and restructured. The dynamically generated pages are equivalent to those in the purely static version of the Web site (they can also be generated offline, to improve performance), with the remarkable advantage that they are ensured to have all the same structure.

The final result is that maintenance of the page specific information is centralized in a database and only one server script has to be modified to correct defects or add new features, instead of an arbitrarily high number of static pages.

2.4 Manual intervention

Up to this point, the whole process of migration to a dynamic site has been fully automated. However, a final intervention of the user may be necessary to refine the template and the database table automatically generated. If template and database include less (or more) information than desired, the user can edit them directly. Alternatively, the user can select a different cluster for *Template extraction*, so as to have fewer, more similar pages (more fields), or additional pages that introduce further constraints (less fields). This is easily achieved by changing the cut point in the hierarchy of clusterings and re-executing *Template extraction* and *Database generation* (both of which are completely automatic).

3. Multilingual pages

The structural edit distance described in the previous section has been used in two different ways: excluding the text between tags (for *Clustering*) or including the text between tags (for *LCS*). In presence of pages in different languages (multilingual Web sites), the comparison of the text between tags cannot be a simple comparison of strings. Two texts in different languages are matched if they are the translation of each other, rather than being exactly the same.

In presence of multilingual pages, it is always possible to split the Web site into mono-lingual portions (*language division phase*) and apply the proposed Toolkit to each sub-site. The language division phase was realized by means of our own algorithm (Tonella *et* al., 2002). The problem of language identification was deeply investigated in the Natural Language Processing research community. We opted for a simple solution that proved effective and efficient in our case. It is based on the entries available in a set of mono-lingual dictionaries, associated with the words in the page to be classified. Our algorithm determines the number of words in the page text that do not have an entry in each dictionary. The dictionary which gives the minimum number of errors (unrecognized words) for the given page is selected. For further information about this algorithm and its performances see (Tonella *et* al., 2002).

Additional benefits in the process are achieved if a unique cross-language template and a unified database are generated for multi-lingual Web sites being restructured. Let us consider the example in Fig.2. Assuming that page *q.html* be an Italian page, instead of an English page, the text within <TITLE> tag and the texts within the two <TH> tags would be translated ("Scheda prodotto", "Nome:" and "Prezzo:" respectively). As a consequence, the extraction of a common cross-language template requires a more sophisticated analysis, exploiting Natural Language Processing (NLP) techniques. More details on this technique are provided elsewhere (Tonella *et* al., 2002). The basic idea is to compute an index (called TCI, Translation Correspondence Index), between 0 and 1, measuring the likelihood that a text be the translation of another text:

TCI = content words with translation / total number of content words

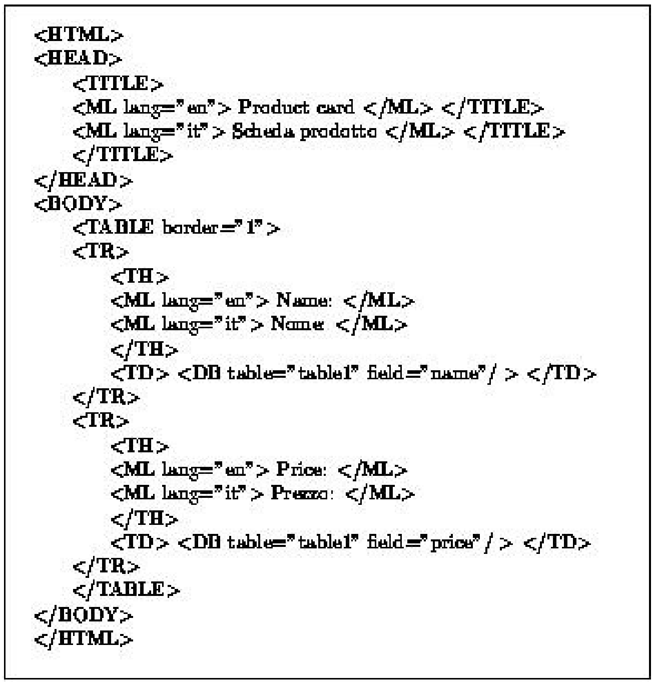
Its computation is based on the entries found in a bilingual electronic dictionary. If the TCI exceeds a given threshold (currently set to 0.3, as determined experimentally), the two texts are considered matching.

In order to generate a cross-language template and database, the Web site is first partitioned according to the language of each page. Then, clustering is used within each language to extract mono-lingual templates. Finally, templates are merged into a common representation, using the TCI values to determine the matching portions.

For the purpose of representing the cross-language template, a possible choice is the adoption of MLHTML (Tonella *et* al., 2002). MLHTML (all tools developed to support migration toward and development in MLHTML are available from the Web site: http://star.itc.it) is a slight extension of XHTML (the new version of HTML compliant with XML, promoted by the W3C consortium) which adds the tag, ML, to the language. When a language-dependent content is embedded in a page, it is surrounded by an ML tag, with an attribute *lang* specifying the language.

An alternative to MLHTML consists of using the database also for those parts of the template which are fixed within a language, but vary from language to language. A database table can be generated storing the language-dependent, but page-independent, parts of the template. A single template is thus obtained, where multilingual texts are replaced by DB tags pointing to proper records of this newly generated database table.

The server side script that constructs the pages dynamically has an additional parameter (*lang*) specifying the selected language. Values are retrieved from the database tables in the language indicated by such a parameter.



```
<HTML>
<HEAD>
    <TITLE>
    <ML lang="en"> Product card </ML> </TITLE>
    <ML lang="it"> Scheda prodotto </ML> </TITLE>
    </TITLE>
</HEAD>
<BODY>
    <TABLE border="1">
    <TR>
        <TH>
        <ML lang="en"> Name: </ML>
        <ML lang="it"> Nome: </ML>
        </TH>
        <TD> <DB table="table1" field="name"/ > </TD>
    </TR>
    <TR>
        <TH>
        <ML lang="en"> Price: </ML>
        <ML lang="it"> Prezzo: </ML>
        </TH>
        <TD> <DB table="table1" field="price" / > </TD>
    </TR>
    </TABLE>
</BODY>
</HTML>
```

Figure 6. Example of MLHTML template obtained as the LCS from the pages in Fig.1, with q.html in Italian

Fig. 6 shows the MLHTML template for the pages in Fig.3 (assuming q.html translated in Italian), as produced by the LCS algorithm. Each multilingual text that is part of the template (i.e., is replicated unchanged in all pages within each language) is surrounded by an ML tag with the specification of the related language. This is the case of the text within TITLE and TH tags. The variable parts of this page are retrieved from the database. They are indicated within a DB tag. Specification of the language is not necessary, since the template is instantiated with a parameter, *lang* holding the selected language. Such a parameter is used also for the selection of the appropriate information, when this is extracted from the database.

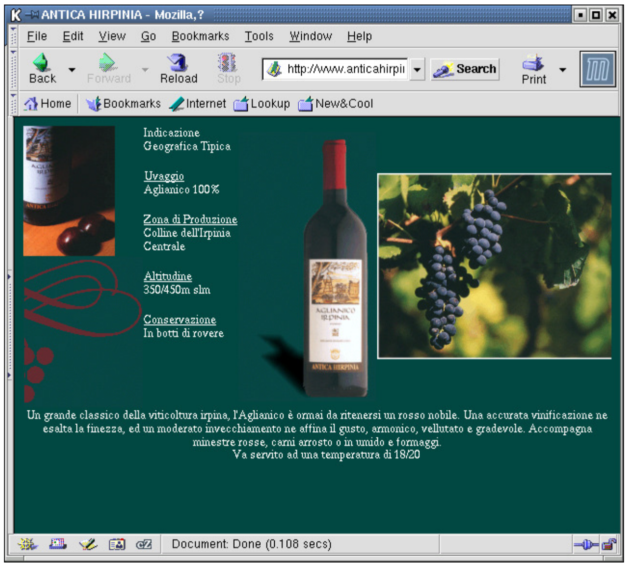
In presence of multilingual pages, the benefits of the proposed restructuring toolkit are even more evident (Tonella *et* al., 2002). In addition to the separation of data from presentation and of the simplification of the source code to maintain, the resulting Web site ensures a consistent presentation of information in all languages. Usage of MLHTML or of a database for the multilingual parts of the template centralizes the maintenance of all different versions for all supported languages.

4. Case Study

A preliminary evaluation (Ricca and Tonella, 2003) of the proposed Toolkit has been conducted on eleven Web sites, chosen among those that seemed to include pages with repeated descriptions of items that are not generated dynamically. The results presented in (Ricca and Tonella, 2003) are good: clustering has revealed itself as an effective method to recognize a common structure of Web pages, the computation of the LCS has provided an automatic way to generate the template to use for dynamic page generation and for each restructured Web site the manual work to be done was definitely reasonable and limited to a few additions and deletions of records, fields and values.

In the remaining of this section, we describe how our Toolkit was applied to a small case study, (*www.anticahirpinia.it*). The operations involved in the restructuring process are considered in detail.

Www.anticahirpinia.it is a bi-lingual Web site of a small wine factory. This Web site, downloaded by means the tool **ReWeb** (Ricca and Tonella, 2001), consists of 36 static HTML pages: 8 HTML pages are Italian wine cards, (an example is provided in Fig. 7), 8 pages are the corresponding cards in English, while the other 20 pages (in English and in Italian) contain descriptive material on the company or provide services (e.g., to order products on-line).


ANTICA HIRPINIA - Mozilla,?
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://www.anticahirpi
Search
Print
Home Bookmarks Internet Lookup New&Cool
Document: Done (0.108 secs)

Indicazione
Geografica Tipica

Uvaggio
Aglianico 100%

Zona di Produzione
Colline dell'Irpinia
Centrale

Altitudine
350/450m slm

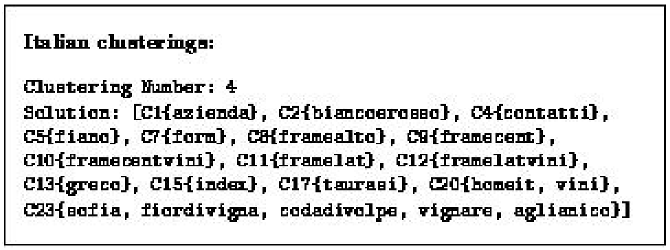
Conservazione
In botti di rovere

Un grande classico della viticoltura irpina, l'Aglianico è ormai da ritenersi un rosso nobile. Una accurata vinificazione ne esalta la finezza, ed un moderato invecchiamento ne affina il gusto, armonico, vellutato e gradevole. Accompagna minestre rosse, carni arrosto o in umido e formaggi.
Va servito ad una temperatura di 18/20

Figure 7. Example of Italian wine card from the Web site www.anticahirpinia.it.

The result of the *language division phase* was 30 correct language classifications out of 36 pages analyzed. Five incorrect and 1 ``Don't Know'' outputs are associated to pages with very little text inside. Considering only cards, the result of language division was 16 correct language classifications out of 16 cards, with no error.

When the clustering algorithm is applied to the Italian portion of the Web site, 15 possible partitions are obtained. Among these, the partition number 4 (cut point equal to 4) has been chosen automatically by the *Cut Point selector* algorithm, because it contains the cluster C23 (see Fig. 8), with five pages (criterion: "min height & #pages >= 4" satisfied).



```
Italian clusterings:

Clustering Number: 4
Solution: [C1{azienda}, C2{biancoerosso}, C4{contatti},
C5{fiano}, C7{form}, C8{framealto}, C9{framecent},
C10{framecentvini}, C11{framelat}, C12{framelatvini},
C13{greco}, C15{index}, C17{taurasi}, C20{homeit, vini},
C23{sofia, fiordivigna, codadivolpe, vignare, aglianico}]
```

Figure 8. Example of clusterings for the Italian portion of www.anticahirpinia.it

By applying the LCS algorithm to the set of pages in C23 the Italian template has been obtained. It contains the following list of potential fields: "Antica Hirpinia", "Uvaggio", "Zona di Produzione", "Altitudine" and "Conservazione". The result of the TXL function *buildFieldPlusValue* applied to these fields for each Italian page is represented in Table 1 (pages with the number of non-null values less than 50% have been removed).

PageId	Antica Hirpinia	Uvaggio	Zona di produzione	Altitudine	Conservazione
aglianico	I.G.T.	Aglianico 100%	Irpinia centrale	350/450m	Botti di rovere
codadivolpe	I.G.T.	Coda di volpe 100%	Irpinia centrale	350/450m	Acciaio inox
Fiano	D.d.O.	Fiano 100%	Avellino e colline	300/400m	Acciaio inox
fiordivigna	I.G.T.	Aglianico 100%	Irpinia centrale	350/450m	Acciaio inox
Greco	D.d.O.	Greco 100%	Tufo, santa paolina	450/600m	Acciaio inox
Sofia	I.G.T.	Fiano 80% Coda di volpe 20%	Irpinia centrale	400/450m	Acciaio inox
Taurasi	D.d.O.	Aglianico 100%	Colli taurasini	350/400m	Botti di rovere
Vignare	I.G.T	Aglianico 100%	Taurasi	360m	Rovere e carati

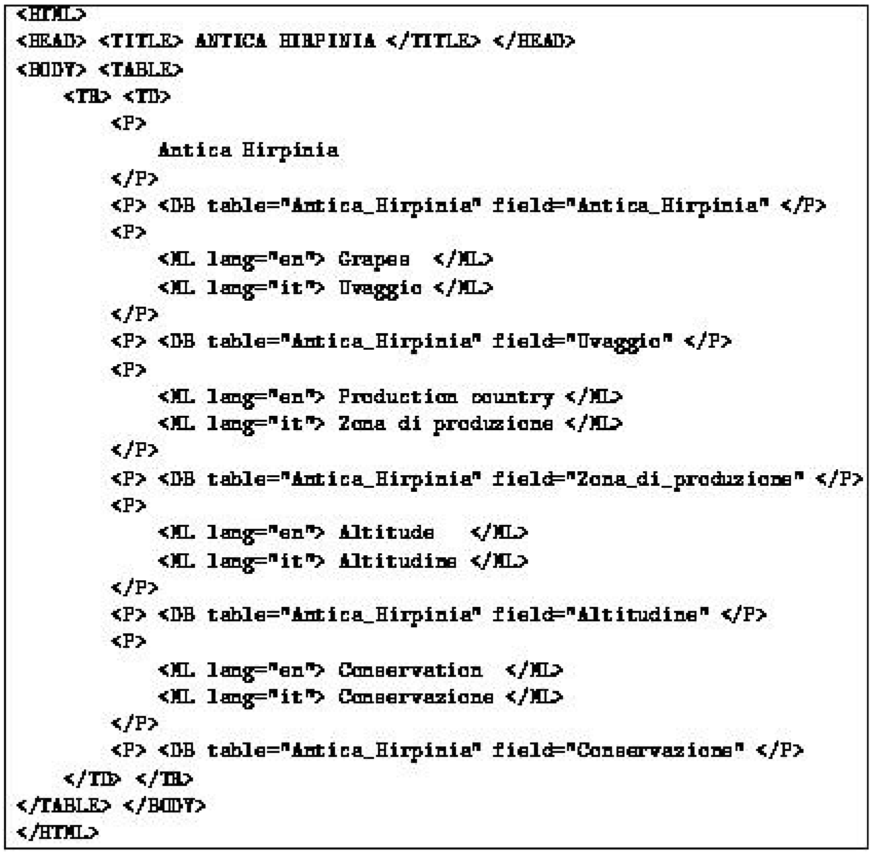
Table 1. The Italian database generated by our Toolkit (I.G.T ="Indicazione Geografica Tipica", D.d.O = "Denominazione di origine")

According to a visual inspection of the Web site, the database generated in this case is almost perfect. The user has just to rename the column *"Antica Hirpinia"* into *"Origine"*. When navigating the new Web site, the server script *T.php*, extracting the information specified in each DB tag from the generated database, is accessed, instead of the initial set of (inconsistent) static HTML pages.

When repeating the restructuring process on the English portion of the site, 11 partitions are obtained and the cluster C17 (cut point equal to 2), with 4 pages, is chosen for template generation. Given the template, a data base containing texts in English was automatically generated. Even in this case, the intervention of the user on the database is minimal, being limited just to renaming a column (again, "Antica Hirpinia").

To obtain additional benefits and a unified database, the Italian and English templates produced by clustering within each mono lingual site portion have been merged into a single, common template.

A (simplified) version of the cross-language template (for readability, images and some tag attributes have been removed) is shown in Fig. 9.



```
<HTML>
<HEAD> <TITLE> ANTICA HIRPINIA </TITLE> </HEAD>
<BODY> <TABLE>
    <TR> <TD>
        <P>
            Antica Hirpinia
        </P>
        <P> <DB table="Antica_Hirpinia" field="Antica_Hirpinia" </P>
        <P>
            <ML lang="en"> Grapes  </ML>
            <ML lang="it"> Uvaggio </ML>
        </P>
        <P> <DB table="Antica_Hirpinia" field="Uvaggio" </P>
        <P>
            <ML lang="en"> Production country </ML>
            <ML lang="it"> Zona di produzione </ML>
        </P>
        <P> <DB table="Antica_Hirpinia" field="Zona_di_produzione" </P>
        <P>
            <ML lang="en"> Altitude   </ML>
            <ML lang="it"> Altitudine </ML>
        </P>
        <P> <DB table="Antica_Hirpinia" field="Altitudine" </P>
        <P>
            <ML lang="en"> Conservation  </ML>
            <ML lang="it"> Conservazione </ML>
        </P>
        <P> <DB table="Antica_Hirpinia" field="Conservazione" </P>
    </TD> </TR>
</TABLE> </BODY>
</HTML>
```

Figure 9. Simplified MLHTML template for www.anticahirpinia.it

5. Related Works

Several works on Web application understanding, restructuring and reengineering have been presented in literature.

An experience of Web site reengineering is summarized in (Antoniol *et* al., 2000). This work describes the recovering process of an abstract model of a target Web site and the following re-design and implementation activities, both conducted almost completely manually. The formalism used in the process is RMM (Isakowitz *et* al., 1997).

The work most related to ours is (Estievenart and Francois, 2003). This paper presents a tool-supported method to reengineer static web sites. The tool analyzes all the pages of a Web site in order to derive their common structure (if any). This structure is formalized by an XML document, called META, which is then used to extract an XML document that contains the data of the pages and an XML schema validating these data. A META document can describe various structures such as alternative layout and data structure for the same concept, structure multiplicity and separation between layout and informational content. XML schemas extracted from different page types are integrated and conceptualised into a unique schema describing the domain covered by the whole web site. Finally, this conceptual schema is used to build the database of a renovated Web site.

In (Boldyreff and Kewish, 2001) the authors describe a system to extract some items, such as the textual content or the images, from the Web pages of an existing site and to store them into a database. The original Web pages are recreated by retrieveing the migrated information from the database. Replications of a same information are detected and removed.

Clustering has several applications in software analysis and reverse engineering, and has been recently applied to Web applications (Dilucca *et* al., 2002a) with the aim of supporting program understanding.

Cloned Web pages are identified in (Dilucca *et* al., 2002b) by exploiting an edit distance measure similar to ours, but used for different purposes.

6. Conclusions

Several existing Web sites have not yet moved to the next, dynamic generation of Web applications. Since this kind of Web sites is difficult to evolve (because formatting/style is mixed with content), a restructuring Toolkit targeted to them is potentially very useful.

In this book chapter, we have presented a Toolkit to replace a set of static Web pages having the same structure with a unique server script that generates dynamic pages. The dynamic pages produced on the fly by the server script have a fixed part and a variable part that is built using the information migrated to a database.

The Toolkit consists of three interacting tools. The *Template Extractor tool* recognizes a common structure of Web pages and provides the candidate template to be used in the dynamic generation of the pages, the *Database Generator too*l provides the records to be inserted into the database and the *Dynamic Page Generator tool* builds a script that generates the migrated pages dynamically, from the template and the database.

The proposed approach is expected to improve the maintainability of the restructured Web sites. Infact, page information is stored in a database, while the presentation format is defined in a single place, the template in use, for all the migrated pages. This ensures that the displayed information be consistent across the pages.

For multilingual Web sites the advantages of this approach are even more evident, since these sites have the additional problem of the consistency of structure and content across different languages. Having a unique MLHTML template (or a centralized database) for all supported languages alleviates the consistency problem and simplifies Web site evolution.

In this chapter a working session with our Toolkit has been presented. A real-world, small Web site has been restructured, migrating 16 static pages into a database. When navigating the new restructured Web application, the referenced server script generates consistent pages, with the same structure, style and formatting, even when switching from a language to another one. On the contrary, the initial set of static HTML pages contained several inconsistencies.

Bibliography

Anquetil N., Lethbridge T.C. (1999), “Experiments with clustering as a software remodularization method”, *Proceedings of the 6th Working Conference on Reverse Engineering,* IEEE Computer Society Press, pp. 235-235

Antoniol G., Canfora G., Casazza G., De Lucia A. (2000). “ Web site reengineering using RMM”. *Proceedings of 2th International Workshop on Web Site Evolution*. Zurich, Switzerland, pp. 9-16.

Atallah M.J., (1999), “Algorithms and Theory of Computation Handbook”, CRC Press, Boca Raton, Florida, USA.

Cordy J.R., Dean T.R., Malton A.J., Schneider K.A. (2002), “Source Transformation in Software Engineering using the TXL Transformation System”, *Information and Software Technology* 44 *(*13)*:*827-837

Cordy J.R, (2004) "TXL - A Language for Programming Language Tools and Applications", *Proceedings of LDTA 2004, ACM 4th International Workshop on Language Descriptions, Tools and Applications*, Barcelona.

Di Lucca G.A., Fasolino A.R., De Carlini U.D., Pace F. Tramontana P. (2002), "Comprehending Web applications by a clustering based approach", *Proceedings of the 10th International Workshop on Program Comprehension*, Paris, France, IEEE Computer Society, pp. 261-270

Di Lucca G.A., Di Penta M., Fasolino A.R., (2002), "An approach to identify duplicated Web Pages", *Proceedings of the 26th Annual International Computer Softawre and Applications Conference, Oxford, England,* IEEE Computer Society, pp. 481-486

Estievenart F., Francois A. (2002). "A tool-supported method to extract data and schema from web sites", *Proceedings of 5th International Workshop on Web Site Evolution*, Amsterdam, The Netherlands.

Isakowitz, T., Kamis, A., Koufar, M. (1997). "Extending RMM: Russian Dolls and Hypertext". *Proceedings of HICSS-30.*

Ricca F., Tonella, P. (2001). "Analysis and Testing of Web application", *Proceedings of the International Conference on Software Engineering, Toronto*, Ontario, Canada, pp. 24-34

Ricca, F., Tonella, P. (2003). "Using Clustering to Support the Migration from Static to Dynamic Web Pages*", Proceedings of the International Workshop on Program Comprehension*, Portland, Oregon, USA, pp 207-216.

Tonella, P., Ricca, F., Pianta, E., Girardi, C. (2002). "Restructuring Multilingual Web Sites*", Proceedings of the International Conference on Software Maintenance, Montreal*, Canada, October, pp 290-299.

Wiggerts T. (1997), "Using clustering algorithms in legacy systems remodularization", *Proceedings of the 4th Working Conference on Reverse Engineering*, IEEE Computer Society, pp. 33-43