

# Web Reverse Engineering

Fabrice Estiévenart<sup>1</sup>, Aurore François<sup>1</sup>, Jean Henrard<sup>1,2</sup>, Jean-Luc Hainaut<sup>2</sup>

*CETIC, rue Clément Ader, 8 - B6041 Gosselies - Belgium (1)*

*Institut d'Informatique, University of Namur, rue Grandgagnage, 21 - B5000 Namur - Belgium (2)*

*{fe, af, jh}@cetic.be, jlh@info.fundp.ac.be*

## Abstract

*Modern technologies allow web sites to be dynamically managed by building pages on-the-fly through scripts that get data from a database. Dissociation of data from layout directives provides easy data update and homogeneous presentation. However, many web sites still are made of static HTML pages in which data and layout information are interleaved. This leads to out-of-date information, inconsistent style and tricky and expensive maintenance.*

*This paper presents a tool supported methodology to reengineer web sites, that is, to extract the page contents as XML documents structured by expressive DTDs or XML Schemas. All the pages that are recognized to express the same application (sub)domain are analyzed in order to derive their common structure. 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 a XML Schema validating these data. The 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, the data are converted according to this new schema so that they can be used to produce the renovated web site. These principles will be illustrated through a case study using the tools that create the META document, extract the data and the XML Schema.*

**keywords:** reengineering, web site, XML, data extraction.

## 1. Introduction

Web sites can be seen as software systems comprising thousands of line of "code" (HTML) split into many "modules". Static HTML pages are sort of programs with encapsulated data, which violate software engineering good practice.

Nowadays large web sites are dynamically managed. The pages are built on-the-fly through (programs) scripts that get data from a database. The dissociation of the data from the layout can overcome or simplify web site maintenance problems [2] such as out-of-date data, inconsistent information or inconsistent style.

Web sites publish a large amount of data that change frequently and the same data are present on many different pages (redundancy). If the data are stored in a well structured database, keeping them up-to-date is easier than if these data were disseminated in a many of pages. Pages that are generated through a script or a style sheet have all the same style which increases the user's comfort and gives a more professional look to the site. If the style of the web site must be changed, only the script or the style sheet need to be changed and all the pages respect this new style.

Another advantage of the separation of the data and their layout is that the same data can be presented according different layout depending of the intended audience. For example, the same data (stored in a database) can be used to produce the intranet, the extranet and some paper brochures.

Many sites are made up of static pages because such pages can be easily created using a variety of tools that can be used by people with little or no formal knowledge of software engineering. When the size of these sites increases and when these sites require frequent update, the webmasters cannot manage these HTML pages any more. Hence the need to migrate static websites to dynamic ones that are easier to maintain.

This paper presents a methodology to extract the data from static HTML pages to migrate them into a database. If we want to separate the data of static pages from their layout, these pages need to be analysed in order to retrieve the encapsulated data and their their semantic structure have to be elicited. For example, we need to know that a page describing a customer contains its name, one or two phone number(s), an address (itself comprising street, city and zip information). When the data structure of each page has been recovered, it can be transformed into a normalized database. Then, the data can be stored into the newly cre-

ated database. The latter can now be used to generate the dynamic pages of the new web site. This database can also be used for other applications such as e-business application, or to generate information on other media. The database can also be connected to the company back-office applications to exchange up-to-date information.

As suggested, a web site exhibits similarities with a software system. Retrieving its data structure is close to recovering the conceptual schema of a legacy database, a process called, in the database community, *database reverse engineering* (DBRE) [3]. DBRE is described as the reverse of database design, thus we can sketch our web site reverse engineering methodology as the reverse of ideal web site design.

A possible method to design a web site is to decide what kind of information has to be displayed and for each kind of information to define the layout of the pages. Then the pages are created (edited) according to this layout. The layout definition and the pages creation is repeated for each kind of information.

The reverse of this design methodology can be expressed as follows: first, we detect the different type of pages (kind of information); then, for each type of pages we retrieve the common structure and layout; thanks to this common structure we extract the data and we integrate and conceptualize the structure of all the type of page as a global schema, that is, the schema of the future database.

The remainder of the paper is organized as follows. In Section 2, we give a short and non-exhaustive presentation of related works in web sites reverse engineering. Section 3 details the steps of the web sites reverse engineer procedure. After this methodological point of view, we shortly specify the META format that allows us to mark the pertinent components of the page types (Section 4). The method is supported by prototype tools that will be described in Section 5. Finally, Section 6 illustrates the method and the tools with a small but representative case study.

## 2. State of the Art

Many researches and developments propose methods and tools to analyse web page structures and layout. However, goals aimed at are various and sometimes totally differ from ours.

By a static analysis of HTML code, VAQUISTA tries to recover the presentation model of web pages in order to facilitate presentation migration to other kinds of user interfaces [4].

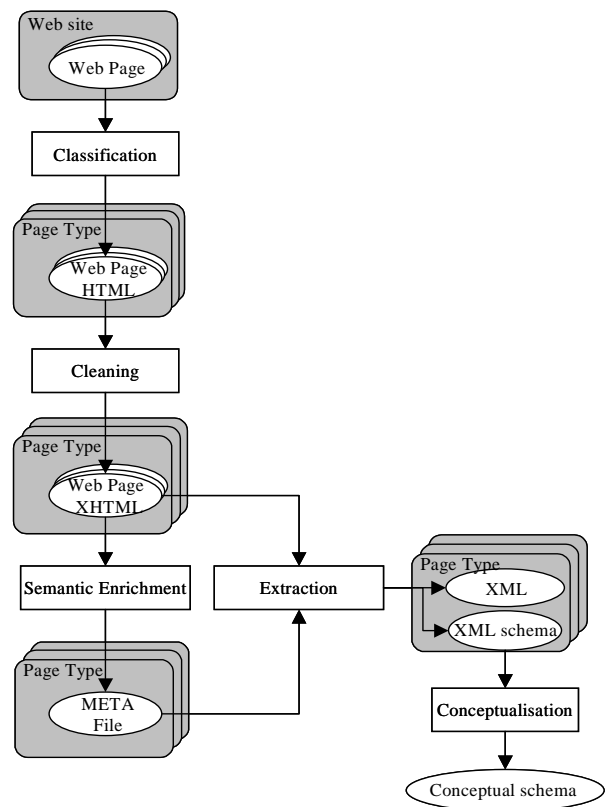
Others claim that by a semantic enrichment of web pages, information can be found on the Web in a more efficient way than by keyword-based queries [5]. In that context, the extracted data are considered as the result of a query rather than the required basis for data migration.

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 [6][7][8][9]. At the opposite, [5] suggests to automate as much as possible that process and therefore assumes that there is some knowledge specific to the topic the web pages are about.

These different approaches use a specific inner formalism to specify the extraction of structures from HTML pages. Lixto [6] is a wrapper generation tool which is well-suited for building HTML/XML wrappers. Patterns discovered by the user are internally expressed by a logic-based declarative language called Elog. In XWRAP [8], the extraction rules are defined in a more procedural way using if...then...else and loop structures. Finally, [7] uses the term "command" to describe a concept, its HTML location and its data value.

## 3. Methodology

### 3.1. Overview of the methodology



**Figure 1. The suggested web reverse-engineering process**

Figure 1 shows a general overview of the proposed methodology. It takes as input data-centric documents [10] in order to produce interpreted XML data and a conceptual

schema covering the domain of the whole web site. "Cleaning" and "Extraction" are two automated processes while "Classification", "Semantic Enrichment" and "Conceptualisation" are user-driven ones. All these processes are chronologically detailed in the next paragraphs.

### 3.2. Web pages classification

Web pages within a site can be gathered into semantic groups according to their informational content. A page type is a set of pages related to a same concept. All the pages of a type have some similarities: they display the same pieces of information and have a very similar, while possibly different, layout. The file path is a good clue to detect the page types through a web site. Indeed, in many cases, all the files belonging to a same type are gathered in the same web directory. For example, `www.myCompany.com/dept/001.htm` and `www.myCompany.com/dept/002.htm` are web pages that describe two departments in a company. They belong to the same type named "Department".

### 3.3. HTML cleaning

When editing web pages with a plain text editor, syntactical mistakes are frequent: opening tags are not closed, quotes around attribute values are forgotten or some element names are undefined in the HTML standards.

HTML interpreters such as web browsers are very laxists when they display HTML pages: visual information could indifferently appear to the user even if unrecognized tags or attributes are used in the source code.

The web reverse engineering process takes as input HTML pages. As HTML is a short-constrained language, it should be useful to check the well-formedness and the validity of these sources before working on it.

Tools, such as Tidy [11], allow to transform any HTML page into a well-formed XHTML-valid document [12]. The result is an XML document that can be parsed and transformed by specific processors such as DOM [13] or XSLT [14].

### 3.4. Semantic enrichment

Defined by the World Wide Web Consortium, HTML is a tagged meta-language used to specify how text, graphics and images will be formatted on a web page. It allows to create headings, item numbered lists, paragraphs and so on. This presentation language is therefore semantically very poor. Identical real world entities can be displayed in many different manners while, at the opposite, information displayed in a homogeneous way do not always have the same meaning. However, heuristics allow to deduce some data

structures from the HTML source code. For example, the headings hierarchy induced by the tags `<H1>`, `<H2>`..., often reflects a tree structure composed by the concepts associated with the headings. Similarly, the lines of a table can be instances of a more general concept associated with the table. Based on these assumptions, the first step in our system redocumentation process will be to add some structure and semantics to the web content. To enable data and structure extraction, information about the name and the position of the concepts in web pages are stored in an XML file called META. A META description is associated with each page type. In the next paragraphs, we detail the format and goal of this description.

#### 3.4.1. Concepts identification and description

As mentioned above, pages of a same type show a similar informational content. Before extracting data and structures from web pages, it is first necessary to identify, among all the pages of the same type, the most significative concepts displayed in them.

In our study, a concept could be defined as a part of a HTML tree describing the layout, the structure and possibly the value of a certain real entity. A concept is therefore not restricted to a value on a web page but is extended to the whole page area related to the described reality.

The following example shows the concept "Phone Number". It consists of a single line (tag `<tr>`) in a HTML table and is therefore composed of both HTML and text nodes.

```
<tr> <td align="middle" bgcolor="#FFFF66">
    <b>Phone :</b>
</td>
<td bgcolor="#FFFF99">
    +32 71 72.23.49
</td>
</tr>
```

The master concept of the page type must be identified. It covers the entire page and gives its name to the page type.

Each concept is characterized by its name and its description, that is made up of an abstract tree containing presentation node(s) (i.e., compliant HTML tags), possibly structure node(s) and value node.

Some concepts include other identified (sub)concepts, providing a hierarchical concept structure that must be materialized in the META file. For example, a concept named "Order" could be a HTML list (tag `<ul>`) whose items (tag `<li>`) form another concept named "Detail". The corresponding HTML code may look like the following:

```
<h1>Order 001562</h1>
<ul>
  <li>1 printer at <b>89 ||</b></li>
  <li>1 flat screen at <b>499 ||</b></li>
</ul>
```

To represent optional and/or multiple occurrences of the same concept in a page, maximum and minimum concept cardinalities are used.

Terminal concepts (leaves), that are not decomposed in subconcepts, may contain the data values. These dynamic pieces of information must be identified in order to make data extraction possible. In the HTML code, a value can be a simple text node, a mixed-content node (text containing HTML tags) or an attribute value.

### 3.4.2. Anchor and links

Hypertext link are a major property of the World Wide Web. As most of the HTML tags are semantically very poor, anchors and links can be interpreted, in many cases, as semantic relations between concepts. All links and anchors within the web site should be marked out as relevant in our reengineering process.

### 3.4.3. Alternative structures

Pages of the same type have similar structure and layout but may exhibit some differences, particularly if HTML code written by hand. We distinguish data structure differences from layout differences. A data structure difference occurs when the data structure of the same (sub)concept varies in two pages of the same type. The concept "Location", for example, can be made up of "Street", "Number" and "City" components on one page while it is expressed by atomic component "Country" on another page.

When the layout of a concept is different on two pages of the same type, it is called a layout difference. In the following example, the label "Name" displayed in the cell of a table appears in bold-face on a page:

```
<tr>
  <td> <b>Name </b> </td>
</tr>
```

and in italic on another page:

```
<tr>
  <td> <i>Name </i> </td>
</tr>
```

Since the META file is to exhaustively describe a page type, it must allow alternative structure to be expressed, that declares multiple descriptions of a unique concept.

## 3.5. Data and schema extraction

When the META file have been completed (i.e., it contains sufficient information to describe all the pages of the same type), it can be used to extract data from HTML pages and to store them in an XML document. The element names and structure of this document are deduced from the semantic information provided by the META file. This data-oriented document has to comply with an XML

Schema [15] that is also automatically extracted from the META file.

## 3.6. Schema integration and conceptualisation

If a site comprises several page types, their XML Schemas must be integrate into a single integrated schema that can then be conceptualized to represent the application domain covered of the whole web site [16].

During this phase, relationships expressed by HTML links are materialised into XML Schema reference mechanisms (xs:key, xs:unique and xs:keyref constraints). These references, similar to database foreign keys, are deduced from data analysis - in this case URLs - and domain expertise.

In practice, the following heuristic is applied. When a page of type X contains a link that targets a page of type Y, we can deduce that a relation exists between the two pages. If throughout all pages of category X, the same type of link targets the same type of anchor on pages of category Y, it appears to be a good candidate to become a foreign key. Of course, only a good knowledge of the domain allows to distinguish semantic relations from purely navigational or external links.

Finally, redundant information is detected and deleted. For example, labels of links often consist in data from the targeted page. Data analysis, especially surrounding links, allows this replicated information to be identified.

## 3.7. Database design and data migration

Classical database engineering process aims at building an information system from the conceptual schema produced in the preceding step [1]. Finally, the extracted XML data are migrated to this new database.

## 4. Syntactic description of the "META" XML file

As mentioned above, the goal of the META file is to provide information about the name, the location and the inner structure of the concepts displayed on the pages of a specific type. This descriptive document will then be used to extract data and their XML structure from web pages.

In the next sections we describe the major syntactic elements of the META file. They consist of reserved XML elements belonging to the namespace "meta".

### 4.1. Root and global elements

For each page type, a list of the displayed concepts has to be drawn. In the META file describing the page type, each concept becomes an element <meta:element> whose

attribute "name" informs about the name of the concept. All these XML elements are globally declared in a XML Schema way, i.e., they are the direct children of the root element, named "HTMLDescription". This root element does not accept other direct child and its first child will be the <meta:element> describing the general concept. Here is the general XML structure of all META file.

```
<HTMLDescription xmlns:meta="http://www.cetic.be/FR/CRAQ-DB.htm" >
  <meta:element name="Main concept">
    ...
  </meta:element>
  <meta:element name="Concept1">
    ...
  </meta:element>
  ...
</HTMLDescription>
```

## 4.2. Subelements and values

The description of a concept is usually composed of its HTML subtree. Some reserved elements have also been defined to represent the concept hierarchy, the concept cardinalities or to locate concept values. The description of a concept is enclosed within the tag <meta:element> of this concept.

The hierarchical concept structure is materialised in the META file through a reference mechanism which allows references to a globally declared concept inside a concept description to be declared. To do so, a <meta:element> with an attribute "ref" is introduced in the HTML tree, replacing the referenced concept. The value of the "ref" attribute is the name of the global referenced <meta:element>. An attribute "min\_card" (resp. "max\_card") can be declared in subelements to specify its minimum (resp. maximum) cardinality within its parent element. If these attributes are omitted, their default value is 1. The example below is a partial description of a concept that contains multiple occurrences of a subconcept.

```
<meta:element name="Concept">
  ...
  <meta:element ref="Subconcept" min_card="1" max_card="1"/>
  ...
</meta:element>
```

Locating concept values inside web pages is necessary to enable data extraction. A value is, usually, a simple text node. In that case, we simply suggest to replace this leaf node by a special XML element called <meta:value/>.

When the value is a mixed-content node, the reserved element <meta:value/> will have an attribute "acceptHTML" set to "yes".

Finally, an HTML element whose attribute "X" represents a relevant data value will be replaced by the element <meta:value/> whose attribute "HTMLAttribute" will be assigned the value "X". The attribute "src" of the HTML

tag <img> is often considered as a relevant data to be extracted. In the META file, a tag <meta:value HTMLAttribute="src"/> replaces the tag <img src=""/>.

## 4.3. Anchors and links

An anchor is a HTML tag <A> having an attribute "name" whose value identifies the tag within the web page. To reflect this property, an element <meta:key> will enclose all significant anchors.

In an analogous way, a link is a HTML tag <A> with an attribute named "href". The content of this tag determines how the link will appear on the web page. It is usually either a simple text node or a HTML tag introducing a clickable picture. In both cases, we suggest to surround significant HTML links with an element <meta:keyref>. This semantic tag may contain a <meta:value/> element if the inner content of the <A> tag is a pertinent data value to be extracted. So, the following HTML code:

```
<a href="www.myCompany.com/type/page002.htm">data</a>
```

will be described in the META file in such a way:

```
<meta:keyref>
  <a><meta:value/></a>
</meta:keyref>
```

## 4.4. Alternative structure

To enable multiple description for a single concept, an alternative structure is defined in the META file. This is introduced by a <meta:choice> element whose direct children are all <meta:group> elements. Each of these groupings contains a probable description of the concept. The content of a <meta:group> element is therefore identical to a simple concept description. Detailed syntax for alternative structures will be used in the case study (Section 6).

## 5. Tools

### 5.1. The semantic browser ("META" generator/editor)

Users that wish to reengineer a web site are supposed to have a good knowledge of the HTML language. However, we think that the tool allowing to locate concepts in web pages should offer a visual comfort. For this reason, we are now developing the semantic web browser. The user can interact with it by selecting and naming some areas in web pages. He can also specify concept cardinalities, locate data values and mark off significant links/anchors. On the basis of these user-driven information, a META file can be automatically generated and applied to all the pages of a

type. The next paragraphs detail the main interactions of the user with that tool.

#### **5.1.1. Analyzing a first sample page**

To enrich web content with semantic information, the user has to analyze, for each page type, a sample web document. In this source web page, the user names and locates the most significant concepts by pinpointing their semantic areas in the browser. There are no constraints on the names of the HTML tags belonging to an area and an area can be included within another.

After processing a first sample file, a first version of the META file is automatically generated. It contains all the semantic information about the name and the location of the concepts in that page.

#### **5.1.2. Applying the META file to other web pages of the same type**

When the user has highlighted all the most important concepts within a page, the HTML grammar defined in the META file for this first page is tentatively applied on all the pages of the type. If a grammar conflict occurs, the system stops the treatment and asks the user to resolve it by adding a new alternative or by modifying a cardinality. The META file is then updated to reflect the structure and the presentation of the newly analysed page.

#### **5.1.3. Iterate for each page type**

The sample page analysis and the application of the pattern on other pages must be performed for each page type. Ultimately, each page type is described by a META file.

### **5.2. The HTML validator**

Usually the semantic browser is used to analyse some of the pages of a type, analyse all of them would take too much time. So this first analysis produces a META file that must be validated against all the pages of that type. The HTML validator checks if all the pages of a given type complies with this META file, i.e., if the latter correctly describes this web page.

To achieve this, a Java program parses in parallel the META file and the web page. The names of the HTML nodes are compared against those declared in the META file while constraints, such as elements structure and cardinalities, are checked out.

If the validation process succeeds for all the pages of a type, the META file is said "exhaustive" and the extraction process may be executed for that type.

### **5.3. The data and schema extractors**

Using the information contained in the exhaustive META file, the data values can be identified in the web pages and can be given specific semantics through expressive names. From these pages and their META file, the data extractor generates a data-oriented XML document. It contains all the extracted data values enclosed in elements. Element names depend on concept names previously defined by the user.

An exhaustive META file describes the layout and the concept structure of all the web pages of a type. By ignoring the presentation elements (HTML tags), the schema extractor extracts the XML Schema that validates the XML document comprising the data. The W3C XML Schema standard was chosen for its precise definition of cardinalities and referential constraints.

Both data and schema extractors are Java programs using the DOM.

### **5.4. The DB-Main CASE tool**

DB-Main is a general-purpose database engineering CASE environment that offers sophisticated reverse engineering tool-sets. Its purpose is to help the analyst in the design, reverse engineering, migration, maintenance and evolution of database applications. DB-Main offers the usual CASE functions, such as database schema creation, management, visualization, validation, transformation, as well as code and report generation. Further detail can be found in [17] and [18].

XML schemas generated in the previous step are imported in DB-Main where they are visually displayed as tree structures. Using advanced transformation primitives, these schemas are integrated and conceptualised.

## **6. Case study**

We will illustrate our purpose by resolving a small case study. A university has decided to reverse engineer its static website made up of pages describing departments and people working in these departments.

### **6.1. Preparation work: classification and cleaning of the pages**

The implementation of the first step - the web pages classification - will confirm our first feeling. Indeed, the analysis of file paths and pages brings us to the conclusion that two pages types are coexisting on the web site: "Department" and "Staff". The second phase, consisting in the cleaning of the HTML code, turns out to be quite easy, thanks to the use of tools like Tidy.

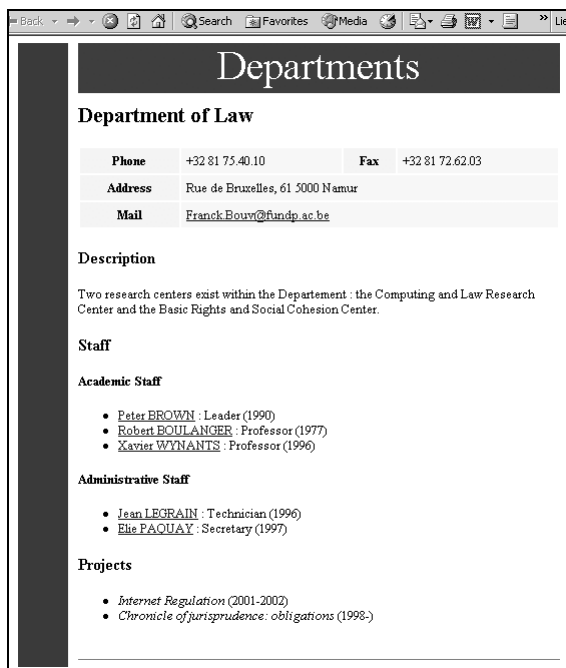


Figure 2. The Web Page of the Department of Law

## 6.2. The page type "Department"

For the needs of this case study, we will mainly examine two pages (Figure 2 and Figure 3) of the Department page type. These pages describe respectively the departments of Pharmacy and Law. One glance at the two pages is enough to ensure that these pages, even if they both definitely belong to the same type, present some differences, concerning their informational content and they layout as well. Concerning the data structures, we notice that some elements are present in one page but not in the other, while the multiplicity of some elements exhibits variations too. For instance, the page of the Department of Pharmacy (Figure 3) is the only one that contains a map localising the department and to mention the presence of PhD Students, while the page relative to the Department of Law (Figure 2) describes some administrative Staff, which the Department of Pharmacy doesn't seem to be concerned with. In terms of multiplicity, we observe that pages of departments of Law and Pharmacy contain respectively three and two academic staff members and two and three projects. About the layout, we can observe that the staff members are either enumerated in a table (Pharmacy) or introduced by bullets of a list (Law).

Moreover, the names of projects undertaken by Departments of Pharmacy and Law are respectively italicised and written in bold type.

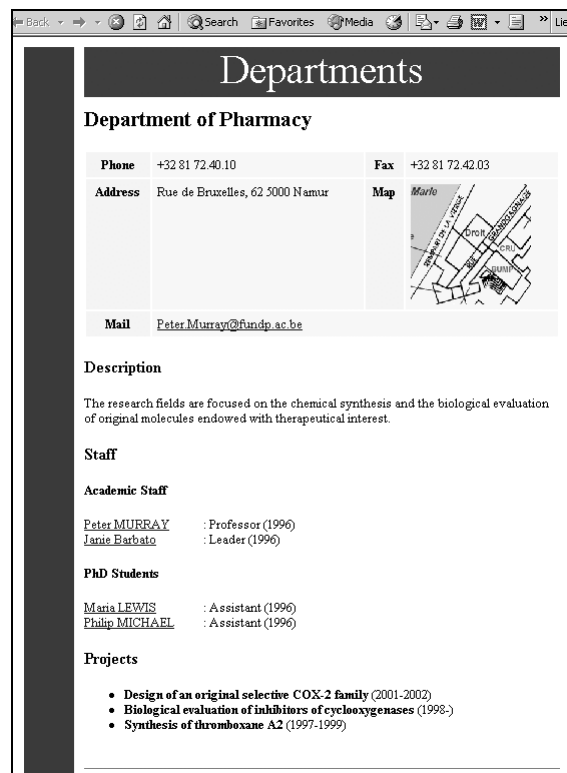


Figure 3. The Web page of the Department of Pharmacy

## 6.3. Building the META file

We construct the META XML document that will inform the extractors about the various structure and presentation of the contents.

We analyse a first sample page, the "Department of Law" page, in which, we locate the concepts of "Department", "DeptName", "Phone", "Fax", "Address", "Mail", "Description", "Staff", "Academic staff", and so on... We all place them in a "meta" namespace and position them as direct children of a <HTMLDescription> root element:

```
<HTMLDescription xmlns:meta="http://www.cetic.be/FR/CRAQ-DB.htm" >
  <meta:element name="Department">
    ...
  </meta:element>
  <meta:element name="DeptName">
    ...
  </meta:element>
  <meta:element name="Details">
    ...
  </meta:element>
  <meta:element name="Phone">
    ...
  </meta:element>
  ...
</HTMLDescription>
```

**Elements definition.** Each `<meta:element>` contains the sequence of HTML tags that describe the concept. The position where the data takes place is marked by a `<meta:value/>` tag. In our example, the META's section describing the "Description" element - that takes place as a paragraph introduced by a third-level heading named "Description" - is the following:

```
<meta:element name="Description">
  <h3>Description</h3>
  <p><meta:value/></p>
</meta:element>
```

**Subelements reference.** Hierarchical structures, like "Details" - that is composed of "Phone", "Fax", "Address", and "Mail" - are represented by elements referenced in their father through a "ref" attribute giving their name. If an element has a multiplicity different from 1-1, "min\_card" and "max\_card" attributes are used.

So, in the META file, the "Details" is represented as:

```
<meta:element name="Details">
  <table> <tr>
    <meta:element ref="Phone"/>
    <meta:element ref="Fax"/>
  </tr>
  <meta:element ref="Address"/>
  <meta:element ref="Mail"/>
</table>
</meta:element>
```

... while the "ProjectsList" element, describing two projects, looks like:

```
<meta:element name="ProjectsList">
  <h3>Projects</h3>
  <ul> <meta:element ref="Project" min_card="2" max_card="2"/> </ul>
</meta:element>
```

Subelements, like "Phone", are completely described further:

```
<meta:element name="Phone">
  <td> <b>Phone</b> </td>
  <td> <meta:value/> </td>
</meta:element>
```

Following this strategy, the META file declares that a Department is composed of the "Name", "Description", "Details", "Staff" and "ProjectList" subelements. A "Staff" element can be divided into "Academic" and "Administrative". Both "Academic" and "Administrative" represent a list of "Person" elements. A "Person" element has two subelements: "PersonName" and "PersonFunction". In turn, "ProjectsList" is composed of "Project" elements, which are declared as sequences of "ProjectName" and "ProjectDates" subelements.

**anchors and links.** `<A>` tags, representing anchors and links, are surrounded by respectively `<meta:key>` and

`<meta:keyref>` elements. The latest contains a `<meta:value/>` if the inner content of the `<A>` tag is pertinent, which is the case of the "PersonName" element:

```
<meta:element name="PersonName">
  <meta:keyref>
    <a> <meta:value/> </a>
  </meta:keyref>
</meta:element>
```

**The second page analysis.** Let us now consider that a first version of the META file is completed: all the elements of the page have been described. We can apply that META file to the second page, "Department of Pharmacy". Of course, some conflicts - detected by the HTML validator - occur, because of the differences observed before. So, the main problem we encounter when reengineering Web sites, typically structure and layout variations, has now to be solved.

Among many others, the "Staff" element is concerned: its subelements vary from one page to another. Indeed, while the "Department of Law" page mentioned a sequence of "Academic" and "Administrative" as subelements of "Staff", the "Staff" element is now composed of "Academic" and "PhD". As mentioned above, these alternative structures are enclosed by `<meta:group>` elements, themselves surrounded by a `<meta:choice>` tag:

```
<meta:element name="Staff">
  <meta:choice>
    <meta:group>
      <h3>Staff</h3>
      <meta:element ref="Academic"/>
      <meta:element ref="Administrative"/>
    </meta:group>
    <meta:group>
      <h3>Staff</h3>
      <meta:element ref="Academic"/>
      <meta:element ref="PHD"/>
    </meta:group>
  </meta:choice>
</meta:element>
```

Another kind of structure variation affects the multiplicity of subelements declared in a `<meta:element>`. For instance, the "Department of Pharmacy" page contains a "Map" element, not present in the previous page. This optional characteristic can be formalised by a "min\_card" attribute set to "0". The declaration of the "Map" element is added in "Details":

```
<meta:element name="Details">
  ...
  <meta:element ref="Address"/>
  <meta:element ref="Map" min_card="0"/>
  ...
</meta:element>
```

The "ProjectName" element has two kinds of layout: italicised in the case of the Department of Law, it is now

written in bold characters. The reasoning we applied for the "Staff" element can also be used to represent layout differences:

```

<meta:element name="ProjectName">
  <meta:choice>
    <meta:group>
      <i><meta:value/></i>
    </meta:group>
    <meta:group>
      <b><meta:value/></b>
    </meta:group>
  </meta:choice>
</meta:element>

```

All the descendants of the "Staff" element - "Academic", "Administrative", "PhD", "Person", "PersonName" and "PersonFunction" - are expressed in a similar way. This is due to a major variation of layout (HTML Table and List), that has consequences on all levels of the hierarchy.

### 6.4. Data and schema extraction

When the META file has been completed, the data and schema extractors can be run. This automated process produces an XML file containing the data from the two pages and a XML Schema validating this data file.

### 6.5. Schema integration and conceptualisation

The previous processes have to be iterated for each page type. In the case of the university web site that we are studying, the pages describing staff members are also analysed. Afterwards, their data and schema can be extracted. This extraction produces an XML-Schema per page type.

All the XML-Schema are imported into the DB-Main CASE tool (Figure 4), where they are integrated. Before to be able to conceptualize the schema we need to recover two kinds of constraints, namely the relationships between elements and location and redundancies or irrelevant information that can be discarded.

anchors and links, respectively formalised by <meta:key> and <meta:keyref> elements, help us to detect relations. For example, the "PersonName" element declared within the page type "Department" contains a <meta:keyref> element. A data analysis shows that this element always targets pages of the category "Staff". Our domain knowledge allows us to consider this relation semantically rich enough to materialize it by the XML Schema referential mechanism (xs:keyref and xs:unique).

Nevertheless, the "PersonName" element extracted from the pages "Department" appears to be the exact copy of another "PersonName" element displayed on each "Staff" page. This redundant information is removed from the

schema. From now on, the "PersonName" of the member of a department can be found thanks to its key value.

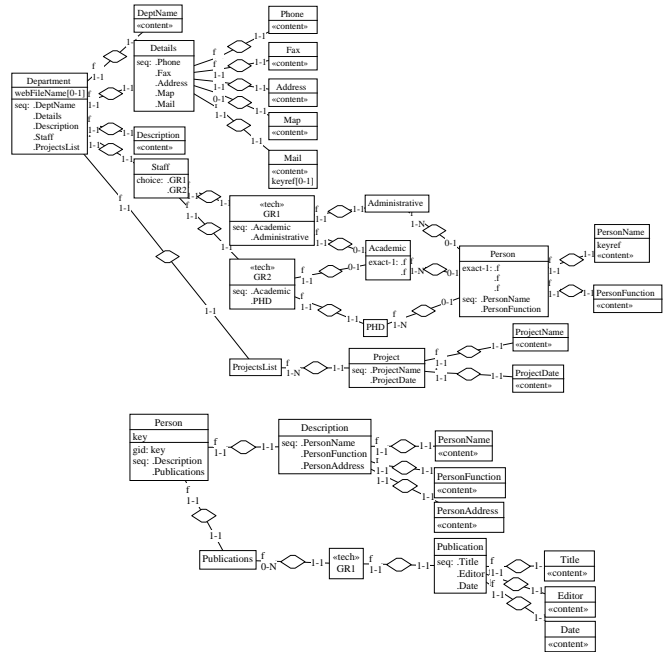


Figure 4. The two XML-Schema imported in DB-MAIN.

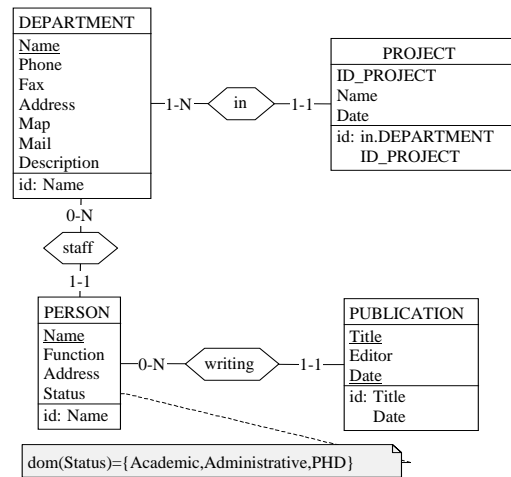


Figure 5. The conceptual schema.

Finally, the integrated schema is conceptualised using reverse engineering specific transformations defined in the DB-Main tool (Figure 5).

### 6.6. Web site engineering

The conceptual schema is transformed into a relational schema (Figure 6). This schema is used to create the new database and the data (the XML file) are migrated into it.

The new web site can be build dynamically by querying this database.

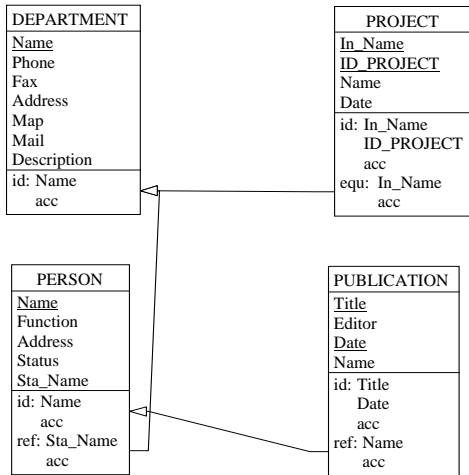


Figure 6. The relational schema of the new database.

## 7. Conclusions and future work

This paper proposes a methodological approach for web sites reverse engineering, supported by tools. It applies on relatively well structured web sites and aims at extracting data values and data structure from the pages. These results are the needed basis for web site redocumentation and data migration towards a newly created information system.

As good practices for web sites engineering and HTML writing are often ignored, we first have to face with an enormous diversity of web pages structures and layouts to represent the same reality. A preparation work made of pages classification and cleaning helps to partially resolve the first difficulty.

A user-driven semantic enrichment is then necessary to make up the relatively poor semantic of HTML code. All the information provided by the user is stored in the META file. That XML document specify, for each page type, the concepts displayed, their location in the HTML tree and their inner description. Using these information, data and their structure can automatically be extracted from all the pages of the analysed web site. This extraction is taken in charge by two java programs that take as input the META file and the HTML pages to produce the XML-schema and the XML data. In a final step, the data structures are integrated and conceptualised to produce a conceptual standardised data schema.

Future work will consist of further tests of the method and XML conventions on real-size web sites. Nevertheless, with the various mechanisms defined in the META file (alternative layout and data structure or structure repetitivity), we think to be able to represent most semi-structured web sites.

With a good knowledge of the conventions, it is always possible to build the META file by hand with a plain XML editor. However, some efforts have still to be done to give the user a more comfortable interface for that task. To achieve this, we are developing a semantic visual browser based on the Mozilla engine. By integrating selected heuristics to the browser, we will achieve a higher degree of automation for the semantic enrichment process. The heuristics would help detect some HTML patterns and automatically deduce the concepts and their implicit structure.

Based on the existing integration primitives included in the DB-MAIN CASE environment, schema integration tools should also be adapted to the XML Schema model.

## 8. References

- [1] Batini, C., Ceri, S., Navathe, S., B.: Conceptual Database Design. Benjamin/ Cummings, 1992.
- [2] Boldyreff C., Kewish R.: "Reverse Engineering to Achieve Maintainable WWW Sites", in Proc. of the 8th Working Conference on Reverse Engineering (WCRE'01), Germany, 2001.
- [3] Hainaut, J.-L., Roland, D., Hick, J.-M., Henrard, J., Englebert, V.: "Database Reverse Engineering: from Requirements to CARE Tools", *Journal of Automated Software Engineering*, 3(1), 1996.
- [4] Vanderdonckt J., Bouillon L., Souchon N., "Flexible Reverse Engineering of Web Pages with VAQUITA", in Proc. of IEEE 8th Working Conference on Reverse Engineering WCRE'2001 (Stuttgart, 2-5 October 2001), IEEE Computer Society Press, Los Alamitos, 2001, pp. 241-248.
- [5] Yip Chung C., Gertz M., Sundaresan N.: "Reverse Engineering for Web Data: From Visual to Semantic Structures", in Proc. of the 18th International Conference on Data Engineering (ICDE'02), 2002.
- [6] Baumgartner R., Flesca S., Gottlob G.: "Visual Web Information Extraction with Lixto", *The VLDB Journal*, 2001.
- [7] Hammer J., Garcia-Molina H., Cho J., Aranha R., Crespo A.: "Extracting Semistructured Information from the Web", in Proc. of the Workshop on Management fo Semistructured Data, 1997.
- [8] Ling Liu, Calton Pu, Wei Han: "XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources", in Proc. of the 16th International Conference on Data Engineering (ICDE'00), 2000.
- [9] Sahuguet A., Azavant F.: "Building Intelligent Web Applications Using Lightweight Wrappers", *Data Knowledge Engineering*, 2000.
- [10] Bourret R.: "XML and Databases", <http://www.rpbourret.com/xml/XMLAndDatabases.htm>, 2003.
- [11] Ragget D.: "Clean up your Web pages with HTML TIDY", <http://www.w3.org/People/Raggett/tidy/>.
- [12] World Wide Web Consortium: "XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)", <http://www.w3.org/TR/xhtml1/>.
- [13] World Wide Web Consortium: "Document Object Model (DOM)", <http://www.w3.org/DOM/>.

- [14] World Wide Web Consortium: "XSL Transformations (XSLT)", <http://www.w3.org/TR/xslt>.
- [15] World Wide Web Consortium: "XML Schema", <http://www.w3.org/XML/Schema>.
- [16] Hainaut J.-L., Tonneau C., Joris M., Chandelon M.: "Schema Transformation Techniques for Database Reverse Engineering", in Proc. 12th Int. Conf. on Entity-Relationship Approach, Arlington-Dallas, E/R Institute Publish., 1993.
- [17] Englebert V., Henrard J., Hick J.-M., Roland D. and Hainaut, J.-L.: "DB-MAIN: un Atelier d'Ingénierie de Bases de Données", Ingénierie des Systèmes d'Information, 4(1), HERMES-AFCET, 1996.
- [18] Hainaut J.-L., Roland D., Hick J.-M., Henrard J. and Englebert V.: "Database Reverse Engineering: from Requirements to CARE Tools", Journal of Automated Software Engineering, 3(1), 1996.