Kitten: a tool for normalizing HTML and extracting its textual content

Mathieu-Henri Falco, Véronique Moriceau, Anne Vilnat

LIMSI-CNRS
University Paris-Sud
91403 Orsay, France
firstname.surname@limsi.fr

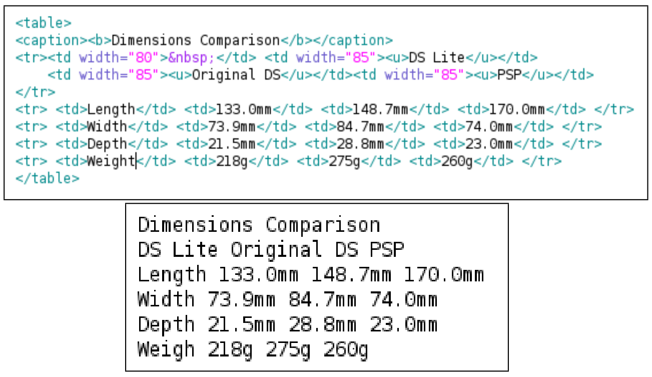
Abstract

The web is composed of a gigantic amount of documents that can be very useful for information extraction systems. Most of them are written in HTML and have to be rendered by an HTML engine in order to display the data they contain on a screen. HTML files thus mix both informational and rendering content. Our goal is to design a tool for informational content extraction. A linear extraction with only a basic filtering of rendering content would not be enough as objects such as lists and tables are linearly coded but need to be read in a non-linear way to be well interpreted. Besides these HTML pages are often incorrectly coded from an HTML point of view and use a segmentation of blocks based on blank space that cannot be transposed in a text file without confusing syntactic parsers. For this purpose, we propose the *Kitten* tool that first normalizes HTML files into unicode XHTML files, then extracts the informational content into a text file with a special processing for sentences, lists and tables.

Keywords: HTML, content extraction, question-answering

1. Introduction

The motivation for coding the *Kitten* tool came during the question-answering system evaluation as part of the QUAERO program as it showed that the data extracted from web pages were confusing the systems based on linguistic features (language model or dependancy grammar notably) (Quintard et al., 2010). The QUAERO corpus is composed of 2 million HTML files (for French and for English) and only a subset is used for the evaluation campaigns: 499,736 files for French and 498,678 for English; from now these subsets will be referred to as *the QUAERO corpus*. Participants were using textual files that had been linearly extracted without any kind of filtering or linguistic processing (Figure 1). It was inadequate mainly because HTML files contain visual blank spaces for segmenting blocks of text where systems expects punctuation marks.

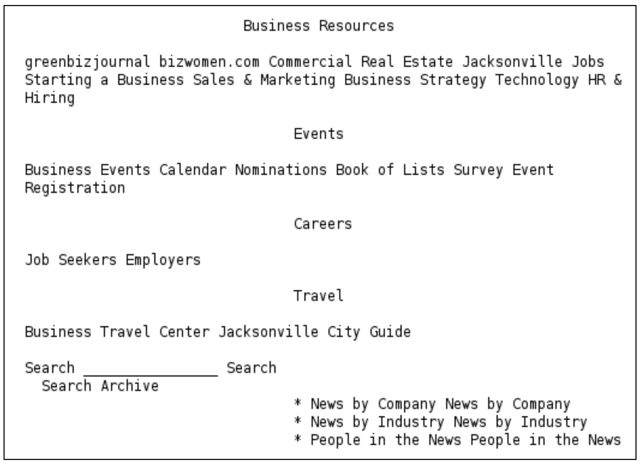


```
<table>
<caption><b>Dimensions Comparison</b></caption>
<tr><td width="80">&nbsp;</td> <td width="85"><u>DS Lite</u></td>
    <td width="85"><u>Original DS</u></td><td width="85"><u>PSP</u></td>
</tr>
<tr> <td>Length</td> <td>133.0mm</td> <td>148.7mm</td> <td>170.0mm</td> </tr>
<tr> <td>Width</td> <td>73.9mm</td> <td>84.7mm</td> <td>74.0mm</td> </tr>
<tr> <td>Depth</td> <td>21.5mm</td> <td>28.8mm</td> <td>23.0mm</td> </tr>
<tr> <td>Weight</td> <td>218g</td> <td>275g</td> <td>260g</td> </tr>
</table>
```

```
Dimensions Comparison
DS Lite Original DS PSP
Length 133.0mm 148.7mm 170.0mm
Width 73.9mm 84.7mm 74.0mm
Depth 21.5mm 28.8mm 23.0mm
Weigh 218g 275g 260g
```

Figure 1: Example of a linear extraction.

A correct extraction must focus on extracting textual content with respect to the limitations of a text file that is read from left to right, line by line and where sentences are delimited by punctuation marks; otherwise texts from different blocks would be merged. The Lynx software allows the dump of an HTML file faithfully to its rendering aspect (Figure 2) but a very difficult segmentational processing has to be done for syntactic parsers to work on correctly delimitated sentences.



```
                          Business Resources

greenbizjournal bizwomen.com Commercial Real Estate Jacksonville Jobs
Starting a Business Sales & Marketing Business Strategy Technology HR &
Hiring

                                Events

Business Events Calendar Nominations Book of Lists Survey Event
Registration

                                Careers

Job Seekers Employers

                                Travel

Business Travel Center Jacksonville City Guide

Search _______________ Search
  Search Archive

                                * News by Company News by Company
                                * News by Industry News by Industry
                                * People in the News People in the News
```

Figure 2: HTML file dumped with Lynx.

In this article, we present the *Kitten* (*Kitten Is a Textual Treatment for Extraction and Normalization*) tool that allows normalization of an HTML file and the extraction of its content into a basic text file. It focuses notably on tables and lists as they need particular attention for their extraction: a simple extraction would not be enough to obtain exploitable textual data for information extraction. *Kitten* is composed of two main steps (Figure 3): after a computational treatment aiming at normalizing the HTML code by combining existing tools (HTMLCleaner[1], jTidy[2] and jChardet[3]), a linguistic processing takes place for extraction

[1]http://htmlcleaner.sourceforge.net/
[2]http://jtidy.sourceforge.net/
[3]http://jchardet.sourceforge.net/

of the informational content into a basic text file. Producing a file at each step makes the use of these existing tools easier. We were not aiming at a spam detector, a removal of boilerplate text (Kohlschütter et al., 2010) or a web page cleaner (Baroni et al., 2008). Our first objective of normalization is very close to the work of Beautifulsoup[4] but the use of BeautifulSoup on the Quaero corpus showed too many losses of files at the time we tested it (19.11% were not completed with the 3.1.0.1 version in January 2010; no file was lost with the more recent version 3.2.0 of November 2010 but Kitten was already developped at that time).

We wanted to normalize the way an HTML file was coded to correctly extract its informational content (including for the moment SPAM and boiler plate content) and finally adapt it to a text file with sentence boundaries so that syntactic parsers can correctly process it.

Kitten processed both the English and French corpus: we provide here figures for the French one. The examples are written in English: some come from the English corpus and those from the French one have been translated.

Section 2 will present the HTML normalization processing and section 3 will focus on the statistical and linguistic processing during the extraction of informational content. Section 4 will present the evaluation of a question-answering system on the QUAERO corpus preprocessed by Kitten, thus giving an indirect evaluation of Kitten.

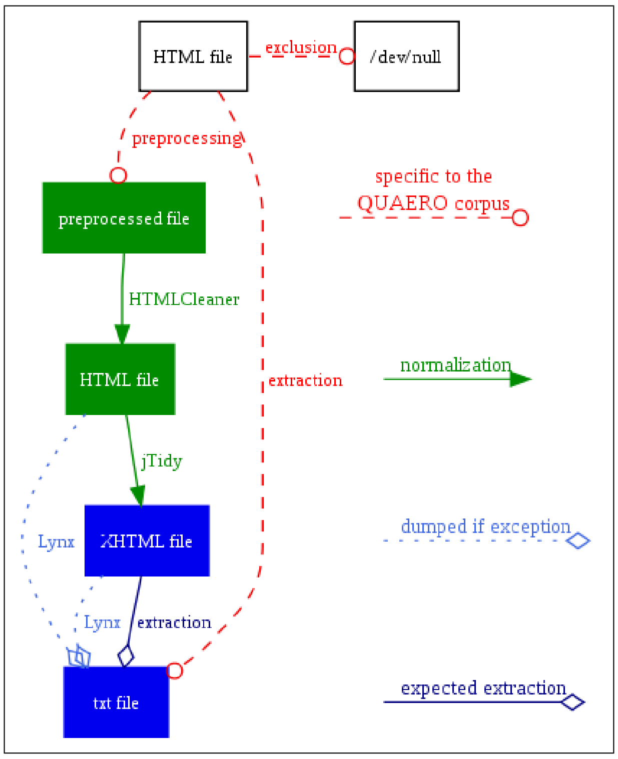


Figure 3: Kitten architecture.

[4]http://www.crummy.com/software/BeautifulSoup

2. From crawled HTML to normalized XHTML

The main goal of the computational processing is to obtain a normalized and UTF-8 encoded XHTML source code. This normalized file must then be XHTML compliant according to the W3 consortium and thus have a valid tree structure for allowing HTML parsing.

2.1. Specific preprocessing

The Kitten architecture (Figure 3) shows three steps: a file exclusion, a direct extraction and a preprocessing. The file exclusion rejects sitemap files as they only contain urls, are very time-consuming and seem to be relevant more for a document classification task (Qi and Davison, 2009) than for a question-answering system. The direct extraction handles already converted binary files (swf, doc, rtf): after having crawled the QUAERO corpus, Exalead[5] converted the binary files into XML files using four tags (*header*, *document*, *textfield*, *page*); these XML files are thus ready to be extracted and do not need to be preprocessed. Almost all files of the QUAERO corpus are preprocessed (99,50%) while only 0,003% sitemap files are excluded and 0.5% XML files directly extracted.

2.2. Fixing HTML errors

HTMLCleaner and jTidy share a lot of common options for fixing HTML errors but we proceeded to several experiments that showed it was better to use them sequentially. HTMLCleaner aims notably at:

- replacing the *&apos;* entity by its character *'* (simple quote): its use is valid in XML and XHTML but not in HTML,
- regarding the content of the tags *<script>* and *<style>* as plain text. It escapes entity like "<" in javascript code that could be confused with an opening tag,
- replacing special HTML entities (i.e. &ocirc;, &permil;) with unicode characters they represent (ô, ‰),

jTidy also aims notably at:

- fixing incorrect tag sequence: for example, <b> <i> </b> </i> becomes <b> <i> </i> </b>,
- closing unclosed tags,
- setting the indentation length within the source file to zero (as it would otherwise insert new carriage-returns inside textual content that will cut sentences).

2.3. Normalizing the files

By fixing incorrectly ordered or non-closed tags, jTidy also allows the normalization of HTML files into XHTML. We use jChardet[6] to detect the most likely encoding of the HTML file allowing thus jTidy to produce an XHTML file encoded in UTF-8.

[5]http://www.exalead.com/search/: one of the QUAERO participant

[6]java port of the source from Mozilla automatic charset

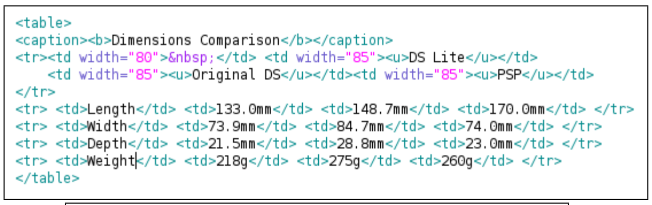
Both HTMLCleaner and jTidy work on transformation of entities in order to return an unicode file. Non-ascii characters can be encoded on the web in three different ways: for example, the *numeric entity* &#230;, the *character entity* &aelig; and the *special character* æ represent the same character. As we want to use only numeric entities after jTidy, a final conversion has to be done for the range of numeric entities from &#128; to &#159; as they are invalid in XHTML.
It was impossible to produce a normalized XHTML file for 5 files. In this situation, a textual dump of these files has been produced with the Lynx browser.

3. From normalized XHTML to exploitable text

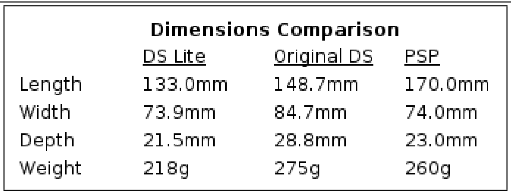
The main goal during the extraction of textual content is to produce a text file that can be read directly by syntactic parsers. By doing so, we had to find a way to transform content using layout (once rendered by an HTML browser) such as tables, lists and blocks of textual content. The examples cited in this section come from needs of Question-Answering applications but all of these features can be set before launching Kitten.

3.1. HTML tables

A table often contains precious informational content and a linear extraction would be particulary inadequate as it is coded linearly in an HTML file, i.e from left to right and row by row (Figure 4) but has to be read by linking semantically a data cell content with its header cell.



```
<table>
<caption><b>Dimensions Comparison</b></caption>
<tr><td width="80">&nbsp;</td> <td width="85"><u>DS Lite</u></td>
    <td width="85"><u>Original DS</u></td><td width="85"><u>PSP</u></td>
</tr>
<tr> <td>Length</td> <td>133.0mm</td> <td>148.7mm</td> <td>170.0mm</td> </tr>
<tr> <td>Width</td> <td>73.9mm</td> <td>84.7mm</td> <td>74.0mm</td> </tr>
<tr> <td>Depth</td> <td>21.5mm</td> <td>28.8mm</td> <td>23.0mm</td> </tr>
<tr> <td>Weight</td> <td>218g</td> <td>275g</td> <td>260g</td> </tr>
</table>
```



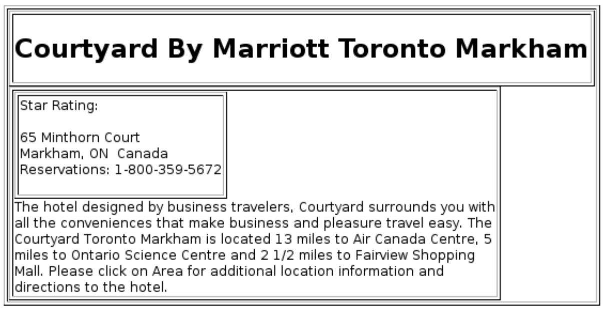
Dimensions Comparison

	DS Lite	Original DS	PSP
Length	133.0mm	148.7mm	170.0mm
Width	73.9mm	84.7mm	74.0mm
Depth	21.5mm	28.8mm	23.0mm
Weight	218g	275g	260g

Figure 4: A data leaf-table: source code and once rendered.

3.1.1. Detecting data tables from layout tables

Kitten targets only leaf-tables: a leaf-table is a table tag not containing itself a table tag. Websites can use recursively the table tag to display information on a page with a depth of 10 nested tables so we decided to focus only on leaf-tables (the non-leaf-table are linearly extracted for the moment). A leaf-table can be a data table or a table used for layout (Figure 5). As this information is not explicitly coded, we use a classifier: a decision tree was built using the machine learning sofware collection WEKA (Hall et al., 2009) based on the features used by (Wang and Hu, 2002) and some others that we added :



Courtyard By Marriott Toronto Markham

Star Rating:

65 Minthorn Court
Markham, ON Canada
Reservations: 1-800-359-5672

The hotel designed by business travelers, Courtyard surrounds you with all the conveniences that make business and pleasure travel easy. The Courtyard Toronto Markham is located 13 miles to Air Canada Centre, 5 miles to Ontario Science Centre and 2 1/2 miles to Fairview Shopping Mall. Please click on Area for additional location information and directions to the hotel.

Figure 5: Layout table.

- rate of empty cases for a line and a row,
- has a <form> tag,
- has a <caption> tag,
- number of <th> tags,
- is a leaf of an embedded table.

An annotated corpus composed of 2,850 layout and 638 data leaf-tables is used for the learning phase:

- 549 layout and 281 data leaf-tables come from 118 files of the corpus annotated by (Wang and Hu, 2002) (we kept only one webpage by website). Their annotated corpus is available[7] and is composed of web documents from 2001: we integrated it to reinforce the robustness of our detection,
- 2,301 layout and 357 data leaf-tables come from 294 files of the QUAERO corpus that we annotated,
- finally we randomly chose among the Wang and QUAERO corpus 614 layout leaf-tables to build the model.

Type	Precision	Recall	F-Score
layout	0.979	0.933	**0.947**
data (Kitten)	0.917	0.965	**0.951**
data (Wang)	0.942	0.973	**0.957**

Table 1: Categorization of 614 layout and 614 data leaf-tables.

The first pass with 10 cross-validations on the annotated corpus showed encouraging results (table 1).

3.1.2. Categorizing the cells of data tables

For all the data leaf-tables identified, a second decision tree was built in order to classify each cell among six categories (table 2):

- *data* and *header* are explicit and obligatory,
- *neutral* is an empty cell for rendering purposes (not for missing data),

[7] http://gsl.lab.asu.edu/doc/webtable.html

- *thema* is the caption of a table or information regarding all the cell data within the table,
- *finalImportant* acts like *thema* but for information located at the bottom of the table (legend for example),
- *finalNonImportant* is for duplicated *header* within the table.

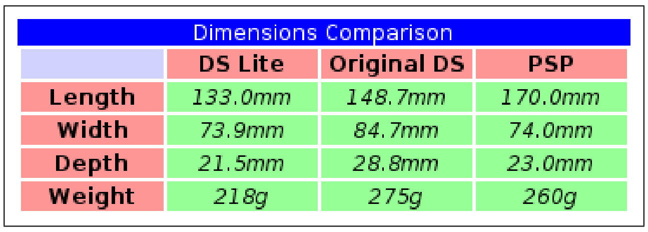
This decision tree was also built with WEKA and also based on the features used by (Wang and Hu, 2002) and some others that we added :

- heigth and length of the cell,
- rate of merged cells on a given line and row,
- number of carriage-returns in the cell.

Once cells are typed, we use manual heuristics to fix some incorrect identification like, for example, a cell identified as *data* while all the others on the same line were identified as *header*.

Class (nb of instances)	**Prec.**	**Rec.**	**F-score**
header (8,542)	0.975	0.967	**0.971**
data (60,648)	0.995	0.997	**0.996**
neutral (4,468)	0.975	0.966	**0.971**
finalImportant (140)	0.897	0.732	**0.806**
finalNonImportant (34)	0.889	1	**0.941**
thema (254)	0.874	0.867	**0.871**

Table 2: Precision, recall and F-score for the identification of cells for the 614 data tables (10-fold cross validation).



Dimensions Comparison			
	DS Lite	Original DS	PSP
Length	133.0mm	148.7mm	170.0mm
Width	73.9mm	84.7mm	74.0mm
Depth	21.5mm	28.8mm	23.0mm
Weight	218g	275g	260g

Figure 6: Table example with correctly identified cells: **header**, *data*, thema (top case), neutral (empty).

Our goal is to extract the table content in order to build a clause which can be parsed by information extraction systems. We combine each data cell content on the same row with its header cell(s) in the same sentence by using the following pattern (the pattern uses regexp symbols): thema? ;;? finalImportant? ;;? (header? ;? header ; data /?)+ . Figure 6 shows the data leaf-table correctly identified and its extraction is the following:

- Dimensions comparison ;; **DS Lite** ; **Length**: *133.0mm* / **Original DS** ; **Length**: *148.7mm* / **PSP** ; **Length**: *170.0mm*.
- Dimensions comparison ;; **DS Lite** ; **Width**: *73.9mm* / **Original DS** ; **Width**: *84.7mm* / **PSP** ; **Width**: *74.0mm*.
- Dimensions comparison ;; **DS Lite** ; **Depth**: *21.5mm* / **Original DS** ; **Depth**: *28.8mm* / **PSP** ; **Depth**: *23.0mm*.
- Dimensions comparison ;; **DS Lite** ; **Weight**: *218g* / **Original DS** ; **Weight**: *275g* / **PSP** ; **Weight**: *260g*.

3.2. Lists

List objects can also confuse syntactic parsers as the items can be clauses syntactically related to the introductory clause (Gala, 2003), (Aït-Mokhtar et al., 2003). To that purpose, a typology of lists was created and extraction is guided according to rules specific to each type :

- conditional removal of hyperlinks in a hyperlink list,
- conditional removal of item bullets,
- list flattening,
- list flattening and introductory clause duplication for each item.

We only consider lists coded using the expected tag such as <ul> (list), <ol> (ordered list) and <dl> (definition list); hardcoded lists using layout tags such as <br> or <p> are too difficult to identify for the moment. For the flattening operation, we only focus on the lists that are not embedded (only one level) and that have an explicit introductory clause (i.e ending by a colon).

3.2.1. Hyperlinks in a hyperlink list

First we removed particular items from hyperlink lists which are lists composed solely of hyperlinks. They can begin or not by an introductory clause. If an hyperlink item is composed of less than five words, it is removed as it is not likely to contain an answer. Each removal is archived and log exploration showed that on the nearly three million removed items, only 481,115 were different and the top-4 occurrences illustrates the purpose of this removal:

- empty words from menu or forum such as *Home* (30,847 occurrences), *Cite* (21,638),
- empty item used for layout (21,468),
- blog item from a calendar such as *september 2007* (8,442).

3.2.2. Hardwritten bullet removal

Inside a list, bullets are sometimes hardwritten by developers and need to be removed for not confusing the parser. A couple of regular expressions is used on the most common such as *, -, *1)*, *a)* .

3.2.3. List flattening

List flattening removes the carriage-return layout by putting each item in the same sentence than the introductory clause in two ways :

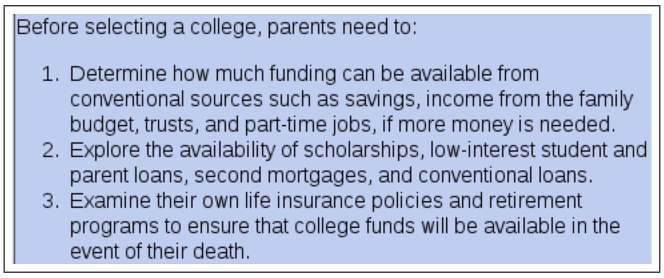
The following list contains a general guideline of different body styles and wedding dress styles to consider:

- *Hourglass-shaped brides*
- *Pear-shaped brides*
- *Petite brides*
- *Plus-size brides*
- *Tall brides*

Figure 7: Example of a list where a linear extraction would move away the item from the introductory clause.

- if the median length of (non-empty) item is lower than 60 characters, the list is flatten to a unique sentence; if an allowed punctuation mark (full stop, question mark, exclamation mark, semi-colon, comma) does not end an item, a comma is added,
- if the median length is greater than 60 characters, we consider that each item should be a sentence so we add a full stop at the end ot the item (when a final punctuation mark is missing) or replace any non-ending punctuation mark by a full stop (except brackets and dash).

For example, the list on Figure 3.2.3. is flattened into the following sentence, improving the probability to answer a question about the terminology of wedding dress style : *The following list contains a general guideline of different body styles and wedding dress styles to consider: Hourglass-shaped brides, Pear-shaped brides, Petite brides, Plus-size brides, Tall brides.*


Before selecting a college, parents need to:

1. Determine how much funding can be available from conventional sources such as savings, income from the family budget, trusts, and part-time jobs, if more money is needed.
2. Explore the availability of scholarships, low-interest student and parent loans, second mortgages, and conventional loans.
3. Examine their own life insurance policies and retirement programs to ensure that college funds will be available in the event of their death.

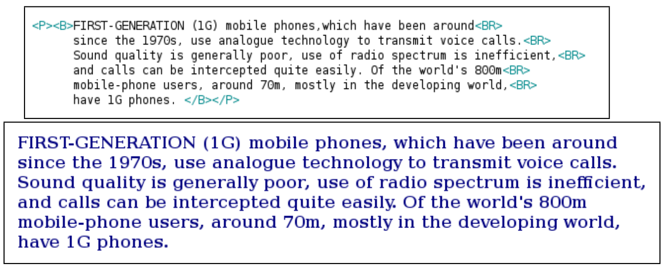
Figure 8: Example of syntactically related items.

Additionally to list flattening, introductory clauses can be repeated with each item (each pair forming a unique sentence) when a trigger word is ending the introductory clause. This happens when a syntactic dependancy links each beginning of the item to the end of the introductory clause. If this clause is not repeated, parsers can be confused as the sentence would not be syntactically correct. The trigger words are thus composed of prepositions (*in*, *to*, ...), auxiliary verbs (*may*, *can*, ...) and the adverb *not*. For example, the list in Figure 8 would be extracted into the following sentences :
Before selecting a college, parents need to determine how much funding can be available from conventional sources such as savings, income from the family budget, trusts, and part-time jobs, if more money is needed. Before selecting a college, parents need to explore the availability of scholarships, low-interest student and parent loans, second mortgages, and conventional loans. Before selecting a college, parents need to examine their own life insurance policies and retirement programs to ensure that college funds will be available in the event of their death.

This type of repetition can be a problem as it reduces the number of items following the introductory clause to one: if the introductory clause specifies the number of expected items, the question-answering system should be able to dig around a sentence composed of the merged pair (introductory + item) in order to gather all the mentionned items. For example, this would be the case if the introductory clause of Figure 8 was : *There are **three** important criteria to check out before selecting a college, parents need to:*.

3.3. Segmentation

The basic unit of syntactic parsers is the sentence delimited by punctuation marks. Without them, giant segments of text would be considered as a unique sentence. On the contrary, separated segments of the same sentence written on two consecutive lines would be considered as two sentences. To optimize the syntactic parsing of textual extracted data, we added a full period when missing: for example between a title and a paragraph or between two paragraphs.
We also need to join two parts of a sentence cut with the <br> tag for rendering result as showned in Figure 9. In a case of a linear extraction, each <br> tag would transform the current line (from the source code) into a unique sentence: the linear extraction would then produce six sentences whereas there are only three.


```
<P><B>FIRST-GENERATION (1G) mobile phones,which have been around<BR>
      since the 1970s, use analogue technology to transmit voice calls.<BR>
      Sound quality is generally poor, use of radio spectrum is inefficient,<BR>
      and calls can be intercepted quite easily. Of the world's 800m<BR>
      mobile-phone users, around 70m, mostly in the developing world,<BR>
      have 1G phones. </B></P>
```

FIRST-GENERATION (1G) mobile phones, which have been around since the 1970s, use analogue technology to transmit voice calls. Sound quality is generally poor, use of radio spectrum is inefficient, and calls can be intercepted quite easily. Of the world's 800m mobile-phone users, around 70m, mostly in the developing world, have 1G phones.

Figure 9: Example of cut clauses due to incorrect use of the <br> tag (source code above, linear extraction below).

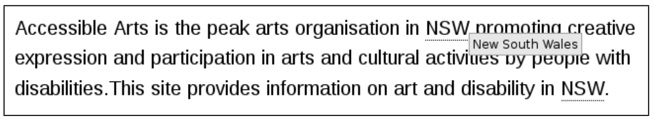

Accessible Arts is the peak arts organisation in NSW promoting creative expression and participation in arts and cultural activities by people with disabilities.This site provides information on art and disability in NSW.

Figure 10: Example of the <acronym> tag that displays informational content *New South Wales* on mouseover near *NSW*.

HTML code contains informational content that would be displayed only with a user interaction like the event "on

mouseover" for <abbr> and <acronym>. If the content is to be extracted, it will be placed between brakets. The extraction of the Figure 10 will be:
Accessible Arts is the peak arts organisation in NSW ***(New South Wales)*** *promoting creative expression and participation in arts and cultural activities by people with disabilities. This site provides information on art and disability in NSW* ***(New South Wales)***.

Around 89,000 abbreviations and 15,000 acronyms were extracted. Logs showed that abbreviation was massively used by a calendar widget of a blog generator: for example, we found 9,318 occurrences of *mon (monday))*. Abbreviation tag also contains sometimes a very precious information for question-answering using temporal inference as a relative temporal expression is associated with an absolute date (e.g. *25 days ago (Wed, 07 May 2008 06:13:06 -0700))*.
On the contrary, some informational content such as the *option selected* attribute from a <form> tag should not be extracted (from a question-answering point of view it should not be extracted as it would generate noise). With the same goal of removing link lists, the tag <div> can be set to be removed if the value of the attribute *id* contains the substring *menu*.

4. Evaluation

A global evaluation of normalization and textual extraction is quite difficult to obtain. We saw that the normalization part was effective but the evaluation depends on the purpose of the extraction. A global evaluation of the question-answering system QAVAL (using a machine learning approach for answer validation) was performed on the QUAERO corpus processed by Kitten and showed a significant improvement (Grappy et al., 2011). Since, Kitten was improved with list flattening and manual heuristics for table cell identification. We then decided to perform a fine-grained evaluation regarding the contribution on tables on one side and lists on the other. For that purpose, we used the FIDJI question-answering system based on syntactic dependencies (Moriceau and Tannier, 2010) hoping a significative gain since FIDJI relies in its core on well formed sentences. FIDJI was evaluated on different processings made on the French Quaero corpus:

- the linear extraction of the corpus done by Exalead ("Baseline"),
- the cleaning of the corpus made by BoilerPipe (Kohlschütter et al., 2010) ("BoilerPipe"),
- a processing of the corpus by Kitten only for tables ("Kitten - Tab"), lists ("Kitten - List") and segmentation ("Kitten - Seg"),
- a complete processing of the corpus by Kitten ("Kitten-Full").

We used the 500 questions of Quaero 2010 evaluation without filtering the question categories which were boolean, definition, factual, list and complex (why, how). We wanted to evaluate the performance of FIDJI on supporting passage extraction (i.e. a passage containing the correct answer). For each question, FIDJI proposes 3 ranked answers with their supporting passage.

Results on table 3 show a gain on the corpora processed by Kitten regarding the baseline (around 24,64% for answers at first rank and 20,18 % for answers at the top-3 ranks) and the runs show that the most significative improvement is given by the segmentation processing.

Corpus	First rank	Top-3 ranks
Baseline	69	109
BoilerPipe	75	116
Kitten-Tab	80	125
Kitten-List	80	123
Kitten-Seg	86	126
Kitten-Full	**86**	**131**

Table 3: FIDJI results for the 500 Quaero 2010 questions: number of questions correctly answered.

5. Conclusion

Kitten is a tool aiming at making exploitable HTML corpus for information extraction. To make HTML corpus exploitable, Kitten integrates:

- existing tools in order to normalize the HTML code and the encoding of the files,
- tools that we developed in order to extract its textual content with a focus on tables, lists, and sentence segmentation.

Textual corpus can thus be processed efficiently by syntactic parsers. Besides, by keeping informational content, Kitten allows web-based question-answering systems to produce better results.
Future works on Kitten will include a module of detecting both SPAM and boilerplate text for removal if asked. It will also focus on improving table header detection (Tajima and Ohnishi, 2008) and multiple header-tab extraction.

Acknowledgement

This work has been partially financed by OSEO under the Quaero program.

6. References

S. Aït-Mokhtar, V. Lux, and E. Bánik. 2003. Linguistic parsing of lists in structured documents. In *Proceedings of the EACL Workshop on Language Technology and the Semantic Web (3rd Workshop on NLP and XML, NLPXML-2003), Budapest, Hungary.*

M. Baroni, F. Chantree, A. Kilgarriff, and S. Sharoff. 2008. Cleaneval: a competition for cleaning web pages. In *Proceedings of the Conference on Language Resources and Evaluation (LREC), Marrakech.*

Nuria Gala. 2003. *Un modèle d'analyseur syntaxique robuste fondé sur la modularité et la lexicalisation de ses grammaires*. Ph.D. thesis, Université Paris-Sud.

Arnaud Grappy, Brigitte Grau, Mathieu-Henri Falco, Anne-Laure Ligozat, Isabelle Robba, and Anne Vilnat. 2011. Selecting answers to questions from web documents by a robust validation process. In *IEEE/WIC/ACM International Conference on Web Intelligence*.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The weka data mining software: An update;. In *SIGKDD Explorations*, volume Volume 11, Issue 1.

C. Kohlschütter, P. Fankhauser, and W. Nejdl. 2010. Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 441–450. ACM.

Véronique Moriceau and Xavier Tannier. 2010. FIDJI: Using Syntax for Validating Answers in Multiple Documents. *Information Retrieval, Special Issue on Focused Information Retrieval*, (10791).

X. Qi and B. D. Davison. 2009. Web page classification: Features and algorithms. In *ACM Computing Surveys, 41(2), February.*

L. Quintard, O. Galibert, G. Adda, B. Grau, D. Laurent, V. Moriceau, S. Rosset, X. Tannier, and A. Vilnat. 2010. Question answering on web data: the qa evaluation in quæro. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10), Valletta, Malta*.

Keishi Tajima and Kaori Ohnishi. 2008. Browsing large html tables on small screens. In *UIST*, pages 259–268.

Y. Wang and J. Hu. 2002. A machine learning based approach for table detection on the web. In *Proceedings of the 11th international conference on World Wide Web*, pages 242–250. ACM.