Georg Rehm

Language-Independent Text Parsing of Arbitrary HTML-Documents. Towards A Foundation For Web Genre Identification

This article describes an approach to parsing and processing arbitrary web pages in order to detect macrostructural objects such as headlines, explicitly- and implicitly-marked lists, and text blocks of different types. The text parser analyses a document by means of several processing stages and inserts the analysis results directly into the DOM tree in the form of XML elements and attributes, so that both the original HTML structure, and the determined macrostructure are available at the same time for secondary processing steps. This text parser is being developed for a novel kind of search engine that aims to classify web pages into web genres so that the search engine user will be able to specify one or more keywords, as well as one or more web genres of the documents to be found.

1 Introduction

This article describes an approach to language-independent text parsing of arbitrary HTML documents. Taking only the HTML element tree and the "visual semantics" (i. e., the canonic way in which a browser renders a certain HTML construct) of elements and attributes into account, we are able to determine macrostructural properties (and in some cases even author-intended functions) of subtrees that will be helpful in several applications that involve processing natural language, especially within the contexts of automatic web genre identification and information extraction.

The main purpose of the text parser described in this article is to abstract from the often highly complex HTML element tree: instead of having to deal with trees containing one node for each HTML element, we want to treat web pages as trees containing only one node for each paragraph, headline, list structure etc. For this purpose, the text parser analyses a document by means of several recursive DOM-based methods that utilize, amongst others, XPath expressions in order to compute features for each subtree. Along with the concrete HTML elements and attributes of a certain subtree, these features can be used to determine its textual function. As soon as one such function is found, the respective subtree is encapsulated by means of a special XML element, the namespace of which is introduced immediately after the initial transformation of arbitrary HTML into well-formed XHTML code. This additional analysis namespace enables a novel view of an HTML document – a view that does not necessarily include every HTML element, but only the macrostructural information that we need in order to be able to tackle the abovementioned processing stages. At the same time, the original HTML element tree is still available in its own namespace for analysis refinement and secondary processing steps.

One of the reasons why it is necessary to analyse HTML documents in the manner proposed in this paper is the phenomenon known as "tag abuse" (Barnard et al., 1996): from a text-structural and text-functional point of view, macrostructural components contained in HTML documents can be *identical* but they can be realized by means of fundamentally *different* HTML elements. It is necessary to reconstruct the macrostructural building blocks intended by the author of a document, as these building blocks cannot be inferred from their respective HTML elements that might be affected by tag abuse (Eiron and McCurley, 2003). This process of reconstructing a document's macrostructure brings us back to the visual semantics of the corresponding HTML elements and attributes. Especially their typographical features need to be examined in order to determine the intended function of a building block. HTML's vocabulary is rather limited, but, ideally, information about navigation bars, headlines, and paragraphs of running text should be available in a consistent set of automatically annotated XML elements.

The remainder of this paper is structured as follows: Section 2 sketches related work on parsing HTML documents. Section 3 introduces the broader context of this article, namely a novel theoretical framework for the automatic identification of web genres. Section 4 describes the text parser, its processing stages, the graphical web front end, and two visualization methods.

2 Related Work

There is a large number of approaches to parsing and processing web pages, driven by objectives as different as the improvement of search and navigation facilities, wrapping, information extraction, automatic summarization, or the adaptation of HTML pages for mobile devices. Carchiolo et al. (2003) parse HTML element trees based on "primary tags" which constitute "primary nodes" (such as `table`, `p`, `map`, `hr`, and `a`). After computing weighted features for each primary node (relative and absolute depth, number of leaves etc.), "collections" are built which are finally mapped onto "logical sections" ("document", "header", "footer", "index", "logical data", and "interactive" sections) defined as "parts of a page each of which collects related information". Yang et al. (2003) present a DOM-based bottom-up approach in order to detect "semantically meaningful clusters", i. e., the implicit schema in template-driven HTML pages. WordNet is employed to find related concepts that can be used as constraints in the structural analysis. Finally, individual clusters are labeled using both statistical and symbolic methods. Chan and Yu (1999) sketch an approach for extracting web design knowledge. First, a "canonical form" is produced based on "primitive" (`br`, `b`, `i`, `font`, `table` etc.) and "compound tags" (`h1`–`h6`, `p`, `ol` etc.). Secondly, "objects" are identified, based on "object tags" and the word count at the leaf level. Three "design knowledge bases" are constructed containing "site layout and navigation", "web page objects", and "web page layouts". Chen et al. (2001) describe the function-based object model (FOM) which was developed for the automatic conversion of HTML documents to WML (Wireless Markup Language). For this process, redundant objects need to be removed and the content needs to be condensed due to the display constraints of mobile devices. With regard to several features of a web page,

Chen et al. (2001) apply statistical methods, decision trees, and visual similarity pattern detection (cf. Song et al., 2004) based on an internal rendering of a page, to determine the specific FOM of individual objects which fall into categories such as information object, navigation object, interaction object, or decoration object. Finally, the non-redundant objects can be used to create a WML-version of the original HTML document.

Myllymaki (2001) describes an information extraction system which is based upon the simple, but powerful notion of processing legacy HTML documents with XML tools. For this purpose, the HTML documents are automatically converted to well-formed XHTML markup, which is, of course, based upon XML, therefore enabling the use of arbitrary XML tools and standards. Myllymaki's ultimate goal is to replicate the databases behind DB-driven web sites. Several key aspects of this wrapping approach can be efficiently carried out by means of cascaded XSLT stylesheets that need to be manually finetuned. Chung et al. (2001, 2002) convert "topic specific" web pages into XML-formats using a DOM-based approach. Examples of topics are "product descriptions", "bibliographies", or "resumés". In the "concept identification" stage, the content of a document is tokenized and matched onto manually predefined concepts. In the subsequent "tree restructuring" phase, the nodes of the intermediate tree structure are rearranged so that "the resulting tree reflects the logical layout of the information carrying objects". In this approach, HTML nodes are gradually replaced by XML element nodes (such as `degree` or `thesis` for the "resume" topic) using the DOM engine. Gupta et al. (2003) present a technique for detecting the content in arbitrary web pages using several filtering stages that process a document's DOM tree recursively, removing and modifying specific nodes, "leaving only the content behind". For example, using the "advertisement remover", certain embedded images whose URLs point to blacklisted web servers can be deleted. By means of the "link list remover", certain link lists contained in table cells can be removed. Gupta et al. (2003) list several potential fields of application for their tool: adaptation of HTML pages for mobile devices, speech rendering for the visually impaired, and summarization. Chen et al. (2003) describe a similar technique with the specific goal of adapting web pages which use the widely employed three-column-layout, to small form-factor displays. The aim of the analysis is to "recover the content structure based on the clues the author embeds in a web page" so that a single web page can be split up into multiple documents that can, in turn, be efficiently browsed on mobile devices. In the DOM-tree based page analysis stage, the "semantic structure" is identified by classifying each node into one of the "content block" categories "header", "footer", "left side bar", "right side bar", and "body". Furthermore, explicit (tags such as `hr`, `table`, `td` and `div`) and implicit boundaries ("blank areas created intentionally") are detected. Afterwards, a document can be split up into multiple documents during the page adaptation stage. Buyukkokten et al. (2001) developed the Power Browser that is able to automatically summarize web pages using an "accordion" metaphor. First, the tool identifies "semantic textual units" within a page and rearranges them into a tree structure that is constructed using the properties of the individual units.

HTML's `table` tag is one of the main instruments in web design. Therefore, several approaches focus upon the automatic processing of tables. Hurst (2002) and Cohen et al.

(2002) try to distinguish genuine tables from tables which are only used to achieve a specific layout, by means of supervised machine learning methods. The features are partially based on a DOM representation and comprise, amongst others, "single HTML row", "single HTML column", and "bag of tags". The authors report a recall of 92–96% and a precision of 95–100% for an evaluation with 89 positive and 250 negative examples. Penn et al. (2001) and Wang and Hu (2002) describe alternative approaches for the detection of genuine tables. Alam et al. (2003) identify tables which are used to mimic lists. Lim and Ng (1999) try to reconstruct the "intended hierarchy of the data content of the table".

3 Genres in the World Wide Web

The Hypnotic project (*Hypertexts and their organisation into a taxonomy by means of intelligent classification*) aims to develop methods and concepts of identifying the respective web genre of arbitrary web pages with the ultimate goal of providing a web genre-enabled search engine (Rehm, 2005). This search engine will ultimately offer a traditional keyword-based search interface augmented with a filtering layer that will give the user the ability to specify the web genre(s) of the documents he or she wants to find (Rehm, 2002). In addition, we see web genre classification as a very promising preprocessing step for novel information extraction tasks (Rehm, 2004a,b).

3.1 Web Genres and Web Genre Modules

The term web genre is a novel concept based upon the classic text linguistics notion of text genre (also known as text type, or text sort). Prominent and extensively researched examples are: *Weather Report*, *Cooking Recipe*, *Letter*, and *Scientific Article*. Some text genres even contain subgeneric variants: A *Love Letter* and a *Business Letter* share a certain number of features that are responsible for the fact that these text genres are both instances of the more general *Letter*, but differ with regard to other features, justifying the assumption that we are dealing with two distinct text genres. Furthermore, text genres can be grouped into multiple hierarchies, e. g., text types that people send to one another (*Letter*, *Fax*, *Text Message*, *Invoice* etc.), or text types in use within academic communities (*Article*, *Memo*, *Technical Report*, *Final Report of a Project*, *Master's Thesis*, or *Ph. D. Thesis*). Traditionally, and on the most general level, text genres can be characterised by the three features *content*, *form*, and *function*. The set of text genres in existence is by no means fixed, but is gradually and unconsciously extended as soon as, for example, a new technology emerges. Mobile phones have introduced a new means of communication, the text message (or SMS, short message service) now being used more than a billion times each day throughout the world. Due to the constraint that one text message must not exceed 160 characters, a novel, heavily abbreviated, telegram-like style of writing has established itself amongst those who use this medium, often borrowing and even re-inventing acronyms and abbreviations that have been reported in computer-mediated communication (CMC) studies on linguistic phenomena within email, Usenet and IRC in

the last ten years (see, e. g., Haase et al., 1997). We do not want to discuss whether text messages can be justified as a new text genre but would like to point out the fact that a newly introduced medium often (and most of the time, inevitably) leads to the emergence of novel text genres, based on new and subsequently recurring communicative situations.

One such area in which new genres develop rapidly, is the World Wide Web. We even go as far as to give this new class of text genres a label of its own: Web genres. World-wide, hundreds of millions of people use the web each day, searching for, reading, and examining web pages. Most of the time, they find the desired information, but, sometimes, the expectations of a user are not met and leave him wondering why certain information is not contained within a specific web page or web site. For example, trying to locate the volume and issue of a certain scientific article needed for a list of references, a user might find the personal home page of the author but a list of his publications may not be contained in his or her web site (cf. Dillon and Gushrowski, 2000). Unfulfilled expectations like these can be explained in terms of the web genre concept: Since the beginning of the 1990s, millions of people have written or developed personal home pages and based the content and layout of their own web pages on the HTML documents they previously encountered. In the early 1990s, these pages were a direct reflection of the possibilities this technological breakthrough offered (Furuta and Marshall, 1996), a distributed hypertext system that allowed people to show off photos of their children, a trip to the Caribbean, or lists of their favourite records. Over the years, the medium came of age, more and more people used the web and, to continue this example, especially the authors of personal home pages intuitively realized that, somehow, certain information does in fact 'belong' on the web while other information does not.

If we consider the *Academic's Personal Home Page*, a subgeneric variant of the *Personal Home Page* (Rehm, 2002), a large variety of information could be found on academics' home pages in the mid-90s, but nowadays there is, as far as content, form, and function are concerned, an almost uniform picture: On their home pages, academics first give their name (often in a large font and accompanied by a photo) and function within the university they work in, they list contact information, current research projects, publications etc. Another web genre is the *Entry Point of a Department* in which, in most cases, lists of staff members, publications, research projects, directions and several other informational units can be found.[1]

Hyperlinks, the most fundamental concept of hypertext, break the boundaries in terms of what constitues a specific web genre. A certain informational unit often contained in a *Personal Home Page* comprises contact information (email address, street address, phone and fax numbers etc.). If this informational unit is embedded within a page, it has a status very different from cases where this information is contained in a file of its own linked to from the home page itself (cf. Mehler et al., 2004). In other words, web genres are by no means monolithic entities, but rather highly modular concepts (see Haas and Grams, 2000): some web genres can only act as web genre instances on their own (e. g., *Personal Home Page*) while others — e. g., the abovementioned *Contact Information* —

[1]Note that, in our theoretical framework, the categories labeled "resume", "product description" etc. by Chung et al. (2001, 2002) are not "topics" but web genres.

can, in addition, be re-used within more general web genres like *Personal Home Page* or *Institutional Home Page*. For this reason, one of our main assumptions is that web genres are composed of web genre modules. For a specific web genre, web genre modules can be either compulsory (e. g., the *List of Publications* in the abovementioned example), or optional (e. g, *Photo Gallery* within *Academic's Personal Home Page*). Furthermore, each web genre has a default assignment with regard to the <content, form, function> triple that is preset by the compulsory modules involved, but which can be modified by the presence of optional modules. Additional details on the relationship between web genres and web genre modules can be found in Rehm (2002), a paper that, furthermore, reviews previous studies with regard to this line of research. As of yet, there are no definitive reports on either the number of web genres or on their hierarchical or taxonomic structure: In most of the related work (see, e. g., Crowston and Williams, 1997, Shepherd and Watters, 1999), random samples of web pages have been drawn using the "surprise" link offered by search engines in the late-90s, i. e., no constraints whatsoever have been employed with regard to the sampling, resulting in very coarse and extremely general lists of dozens of contrasting web genres which are mostly unrelated to one another.

3.2 A Corpus-Database of Web Pages

Due to the abovementioned methodological problems of previous approaches to identifying a coherent set of web genres, we decided to concentrate exclusively on academic web pages: we crawled Germany's academic web (in 2001 and 2002) in order to build a static, and therefore stable corpus of this clearly defined domain, which is perfectly suited for web genre research, without being forced to tackle *all* web genres in existence.

Of the circa 260 German universities (incl. general, technical, specialized and private universities, as well as polytechnics), the corpus contains snapshots of 100 universities. The most important category for our project is "general universities" comprising local mirrors of all the HTTP servers of Germany's 62 'traditional' universities. In total, our crawler traversed 14,968 web servers. Of the 16.2 mio. files visited by the crawler, 4.3 mio. were mirrored in the local filesystem. We further limited the crawl to only those HTML documents written in German, by employing a lexicon-based language identification tool (Rehm, 2001). Table 1 shows the contents of our collection. It can be seen that 3.9 mio. (46%) of the web pages available in Germany's academic web are written in German. Furthermore, a web-accessible PHP- and MySQL-based corpus-database has been developed to facilitate web server and document access and the random generation of document samples. The sample generation can be finetuned with about 20 different parameters, restricting the documents to be randomly included in a sample to certain hostname or filename patterns, domains, file sizes etc. The key components of the corpus-database are an Apache web server which delivers the locally mirrored HTML documents and a MySQL database that contains metadata about these documents, so that efficient retrieval and access methods can be provided. The contents and structure of the HTTP response header (Fielding et al., 1999) acted as a blueprint for the metadata tables (Rehm, 2001) which have not only been populated with the HTTP

Number of universities:	100
• General universities (*complete*)	62
• Technical universities (*complete*)	12
• Music and arts universities (*partial*)	5
• Business universities (*partial*)	5
• Misc. (*partial*)	16
Traversed web servers:	14,968
Web servers operating on port 80:	13,885
Files available via HTTP:	16,196,511
Number of HTML documents:	8,465,105
Total size of all web servers:	701,464.29 MB
Total size of the corpus:	40,914.99 MB
Running word forms (total; `text/html` only):	1,138,794,715
Running word forms (unique; `text/html` only):	12,120,162
Total number of files in the corpus-database:	4,294,417
• Media type `text/html`:	3,956,692
• Media type `text/plain`:	270,400
• Media type `text/css`:	35,651
• Media type `text/xml`:	25,871
• Media type `text/sgml`:	956
• Media type `message/news`:	490
• Media type `message/rfc` :	436

Table 1: Contents of the corpus-database.

response header information of the documents stored on the database-server, but also with the response header data of *all* 16.2 mio. documents and files that were referenced in the original web pages.

3.3 The Document Sample Testbed

The analysis of individual document samples is our primary tool for the identification of specific web genres. After the generation of a sample, the corresponding data set is imported into the corpus-database for future access. Although the sample analysis has to be carried out manually, the database supports the analyser by providing an HTML form of the features to be examined in a pop-up window. After the form for a certain document has been filled out, the information can be saved in the database and later visualized with the analysis data of the other documents in a spread-sheet-like manner.

Several samples have been analysed with the ultimate goal of devising a coherent web genre taxonomy for documents originating within the domain of academia. The examination of an initial sample of 200 web pages was carried out in order to get an initial impression of this domain (Rehm, 2002). Another sample which has been analysed is made up of 727 web pages containing the entry pages of the first 35 universities and the first level of pages linked to from the 35 entry pages. The goal of this analysis was to define the upper structure of the web genre taxonomy (Rehm, 2004b). A third, randomly selected sample of 750 documents has been analysed in order to finalize the leaf level of the taxonomy, which has been modelled in OWL (Rehm, 2005).

Additional samples each contain web pages of specific web genres. One of these samples is used as a testbed for the development phase of the text parser described in the next section, comprises 100 documents of the web genre *Academic's Personal Home Page* and has been collected semi-automatically from web servers offering personal home pages. A list of those pages was randomly shuffled and 100 documents were put into the sample if the web page was (a) the personal home page of an academic, (b) written in German, (c) belonged to one single person (in contrast to, e. g., a research group), (d) primarily dealt with the job of the author at the university and (e) did not use framesets. Table 1 illustrates the (abbreviated) results of this analysis, the primary goals of which were to create a list of the web genre modules involved in this web genre and to determine the status of each module (compulsory vs. optional) based on their individual frequencies within this sample (threshold: 50). The table introduces several new concepts. Web genre modules can be either atomic or complex entities: complex modules consist of two or more features which have to be present in order to instantiate the respective module. Atomic modules consist of only one certain feature. The *Academic's Personal Home Page* is a subgeneric variant of *Personal Home Page*. Therefore, each module in the table can be either general (e. g., *Contact information*) or specific (e. g., *Office hours*) with regard to the more general *Personal Home Page*. Based on an analysis such as the one shown in table 1, we can deduce a definition of the respective web genre in order to specify its content, form and function. Our definition of the *Academic's Personal Home Page*, based on de Saint-Georges (1998), is contained in Rehm (2002).

4 The Text Parser

The previous section introduced our theoretical framework and briefly sketched our ultimate goal of building a web genre-enabled search engine. In order to automatically detect and identify different web genres, we need to take both the content and the structure of a web page into account so that we are able to compute features which can, in turn, be fed to a classification algorithm. A second goal could almost be considered a by-product if the identification of web genres proves to be feasible: If we are able to detect the web genre of a given web page, we could develop means of instantiating the web genre modules involved. Hence, we could extract informational units on a fine-grained level such as the one presented in table 1, in an automatic way. In other words, if we know that a certain HTML document belongs to a certain web genre, we know what kind of content to expect. Based on these expectations, we could try to tackle the information extraction problem (Cowie and Wilks, 2000) on a novel, web genre module-based level.

We developed a text parser for arbitrary web pages in order to lay a foundation for computing features for the web genre classifier, and for the information extraction task. This prototype, implemented in Perl, is embedded in the corpus-database and several means of visualization of the analysis results can be accessed in the "document view" mode by means of the web front end. We developed the parser based on the 100 documents of the *Academic's Personal Home Page* contained in the sample described in section 3.3. The parser's design is based on several principles:

Level	Description	Module/Feature	Status	Number
Atomic	**Explicit greeting**	general	optional	14
Complex	**Identification**	general	compulsory	—
Feature	*Name of the owner of the page*	general	compulsory	100
Feature	*. . . accompanied by an academic title*	specific	compulsory	69
Feature	*. . . accompanied by a job title*	general	optional	27
Feature	*. . . accompanied by an affiliation*	general	optional	34
Feature	*. . . accompanied by a picture of the author*	general	compulsory	54
Complex	**Independent affiliation**	general	compulsory	—
Feature	*Name of the university (in machine readable form)*	general	compulsory	75
Feature	*Logo graphic of the university*	general	optional	16
Atomic	**Alternate version in different language**	general	optional	75
Complex	**Contact information**	general	compulsory	—
Feature	*Street address (university, street, zip, city etc.)*	general	compulsory	90
Feature	*Explicit postal address*	general	optional	8
Feature	*Phone number*	general	compulsory	86
Feature	*Phone number (secretary)*	general	optional	7
Feature	*Fax number*	general	compulsory	66
Feature	*Email address*	general	compulsory	98
Feature	*URL of this home page*	general	optional	4
Feature	*Room/office number*	general	optional	30
Feature	*Send SMS text message*	general	optional	1
Feature	*PGP public key or PGP fingerprint*	general	optional	2
Feature	*X.500 entry*	general	optional	2
Feature	*Directions*	general	optional	2
Feature	*Office hours*	specific	optional	25
Feature	*Address (private)*	general	optional	18
Feature	*Phone number (private)*	general	optional	22
Feature	*Mobile phone number (private)*	general	optional	3
Feature	*Fax number (private)*	general	optional	7
Feature	*Email address (private)*	general	optional	5
Feature	*URL of the private home page*	general	optional	2
Complex	**Contact information (secretary)**	specific	optional	—
Feature	*Name*	general	optional	8
Feature	*Street address*	general	optional	3
Feature	*Room/office number*	general	optional	4
Feature	*Opening hours*	general	optional	5
Feature	*Phone number*	general	optional	6
Feature	*Fax number*	general	optional	6
Feature	*Email address*	general	optional	6
Complex	**Contact information (staff members)**	specific	optional	—
Feature	*Name*	general	optional	7
Feature	*Listing of multiple entries*	meta	optional	6
Feature	*Address*	general	optional	2
Feature	*Room/office number*	general	optional	3
Feature	*Phone number*	general	optional	4
Feature	*Email address*	general	optional	4
Feature	*Names of student assistants*	general	optional	2
Complex	**Academic Profile**	specific	optional	—
Feature	*List of courses*	specific	optional	49
Feature	*Position(s) within the university*	specific	optional	7
Feature	*General student information*	specific	optional	3
Feature	*Suggested titles of final theses*	specific	optional	2
Complex	**Scientific profile**	specific	compulsory	—
Feature	*List of publications*	specific	compulsory	71
Feature	*Research interests*	specific	compulsory	50
Feature	*Research projects*	specific	optional	22
Feature	*Prominently displayed books and/or journals*	specific	optional	6
Feature	*List of talks or presentations*	specific	optional	5
Feature	*Membership in professional associations*	specific	optional	4
Feature	*Technology transfer*	specific	optional	1
Atomic	**C. V.**	general	compulsory	60
Atomic	**Interesting links**	general	optional	12
Complex	**Relevant links**	general	optional	—
Feature	*Link to one's home department/school/insitute*	specific	optional	49
Feature	*Link to one's home university home page*	specific	optional	36
Feature	*Link to one's home faculty home page*	specific	optional	23
Atomic	**Most recent update**	universal	optional	42
Atomic	**Access counter**	universal	optional	11
Atomic	**Guestbook**	universal	optional	1

Table 2: The Web genre *Academic's Personal Home Page*.

Simplicity and robustness We want to keep the algorithms and methods as simple, robust, and general as possible, so that they are, in theory, applicable to all web pages in existence.

Domain- and language-independence We do not want to restrict the functionality of the text parser in any way, i. e., we do not intend to process web pages of a specific domain or language only.

Non-destructive inline analysis annotation We want to keep our analysis results directly within the source document, i. e., we want to keep the original (X)HTML instance and augment this structure with elements and attributes of the analysis namespace (`hypnotic:`) step by step. This principle guarantees that we can access the original data at any point.

Maximum detection of implicit structure We would like to detect as much hidden structure as possible. For example, the authors of web pages tend not to fully exploit HTML's vocabulary, i. e., if an author wants to create a headline, HTML provides the headline-tags `h1` to `h6`, but most authors do not think in terms of tags (logical or structural markup) but rather in terms of font sizes, hence they use physical markup, i. e., the `font` element along with its attribute `size` and a relative or absolute value. Explicit headlines can be detected easily, but, in addition, we also want to find headlines (and sub-headlines) realized by means of the `font` element.

Use of XML standards wherever possible This principle says that we want to use and exploit current XML standards and techniques as much as possible. For example, the Document Object Model (DOM, Hors et al., 2000), XPath (Clark and DeRose, 1999) and XSLT (Clark, 1999) offer several key advantages which enable us to concentrate on the analysis algorithm (DOM, XPath) and the visualization of the analysis results (XSLT).

4.1 Converting HTML into XHTML Markup

The processing of arbitrary HTML documents (which potentially contain markup errors, unknown tags etc.) using XML techniques requires an initial conversion into at least well-formed XHTML code (Myllymaki, 2001). For this HTML to XHTML conversion, a Perl module has been developed which encapsulates the two packages tidy (`http://tidy.sourceforge.net`) and HTML::TreeBuilder (available in CPAN). Using a specific set of configuration parameters, tidy is able to output well-formed XHTML code. Both packages implement rules to cope with invalid HTML structures.

As tidy's rules to correct invalid HTML code are more robust than HTML::TreeBuilder's, our Perl module first tries to pass the source web page through tidy in order to create an XHTML version. If this does not work, we use HTML::TreeBuilder as a fallback tool to repair the HTML input, which is then passed on to tidy again. We evaluated this method by running 10,000 randomly selected documents from the corpus through the module: 98.7% of all documents were successfully transformed to XHTML, the

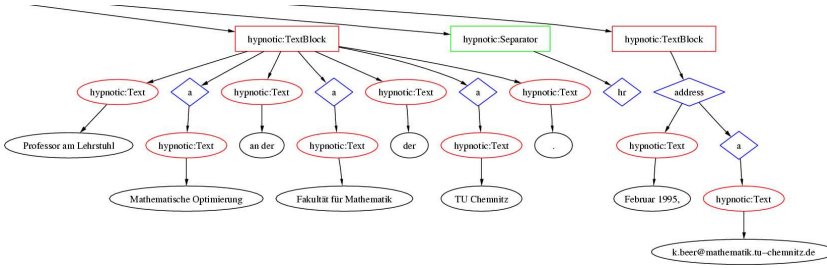

Figure 1: An excerpt from a DOM tree after the automatic insertion of two `hypnotic:TextBlock` elements.

fallback component HTML::TreeBuilder had to be activated 270 times. In addition, we processed these XHTML document instances using the non-validating parser expat (employing XML::Parser). Only 5 of the 9,872 successfully converted documents raised error messages, most of which were caused by character encoding errors. Finally we tried to parse the resulting documents against the XHTML 1.0 Transitional DTD by means of the XML/SGML parser onsgmls. Most of the error messages we found did not relate to malformed markup structures but were rather caused by incorrect attribute values (extremely often in `valign`), or unknown elements (e. g., `blink` or `spacer`).

4.2 The Parsing Algorithm

The parsing algorithm operates in several cascaded stages with increasing complexity and is based on the formal specification of the Hypertext Markup Language as defined in HTML 4.01 (Raggett et al., 1999). The three most important features for the algorithm are those HTML elements which cause a new paragraph to be created while the document is rendered within a browser, a comprehensive analysis of the current font size at any point within the DOM tree, and a rudimentary image analysis. As mentioned earlier, the main goal of this algorithm is to introduce as much explicit structure as possible: we would like to be able to abstract from the often highly complex HTML tree and instead, treat an arbitrary document on a more coarse-grained level.

Stage 1: Basic Annotation of Nodes

This initial stage preprocesses the DOM tree recursively and introduces analysed information as new attributes of our `hypnotic:` namespace within the original XHTML elements.

Each element within the tree receives an attribute `Path` containing an absolute XPath-like path specification (e. g., `/xhtml:html[1]/xhtml:body[1]/xhtml:h1[2]`). Furthermore, links and words (defined as any kind of whitespace-separated token) are counted on both the local and the subtree level. The font analysis computes the current font size for each element, relative to the base font size which can be preset using HTML's `basefont`

```
@s: The sequence of hypnotic:Text elements
@b: Paragraph elements: <p>, <ul>, <ol>, <dl>, <td>, <blockquote>, <div>
$n: The current element
$p: The previous element

while (next element) {
    if ($n == "<hypnotic:Icon>") {
        push(@s, $n);
    }
    if ($n == "<hypnotic:Separator>") {
        markTextBlocks(@s);
    }
    if (($n == "<br>") && ($p == "<br>")) {
        markTextBlocks(@s);
    }
    if ($n is element of @b) {
        markTextBlocks(@s);
    }
    if ($n is not contained in an 'open', partial tree, dominated by a
      node contained in @b) {
        markTextBlocks(@s);
    }
    if ($n == "<hypnotic:Text>") {
        if (!($n->FontSize == $p->FontSize +/-  )) {
            markTextBlocks(@s);
        }
        push(@s, $n);
    }
}
```

Listing 1: `findTextBlocks($r)` in pseudo-Perl code.

element. The base font size is mapped onto the value of 100. Relative font size changes are computed relative to this base. Explicit changes result in absolute values: The headline elements `h1` to `h6` are mapped onto the values 160, 150 etc. The elements `big` and `small` increase or decrease the current size by 10. `strong` and `b` increase the font size by 5 as well, but in the context of a headline element, only by 2. Likewise, `em` and `i` cause an increase of 5. Relative and absolute font size changes realized by means of the `font` element are processed in the same manner (the analysis of CSS information is currently not supported). The image analysis processes files embedded via the `img` element. First, the values of the `src`, `height` and `width` attributes are extracted. Afterwards, the image file is transferred to the analysis machine via HTTP and its dimensions are determined and classified into several categories (explicit `height` and `width` attributes take priority over the physical dimensions): If x and y are less than 6 each, the image is used as a spacer. If x and y are between 6 and 45, we assume the image is an icon. If the quotient $\frac{x}{y} > 10$, the image acts as a separator. If the dimensions of an image are found in a list of quasi-standardised dimensions of banner advertisements (468/60, 156/60, 137/60 etc.), the image is categorised as a banner. The respective `img` element is embedded within one of the following `hypnotic:` elements: `Separator`, `Icon`, `Banner`, `Spacer`.

Stage 2: Text Encapsulation

This stage recursively processes the DOM tree and encapsulates each text node containing one or more words within the element `hypnotic:Text`. Every `Text` node receives several

attributes, especially its font size.

Stage 3: Detection of Text Blocks

The purpose of this stage is to find and to mark text blocks among the `hypnotic:Text` children found in stage 2. A text block is defined as a paragraph-like object in a more or less consistent font size. The function `findTextBlocks` utilizes a top-down/depth-first tree walker that triggers `markTextBlocks` if a text block has been found. Listing 1 shows this function in abbreviated pseudo-Perl code. Several conditions can trigger `markTextBlocks` to be called. These conditions act as boundaries between text blocks: (a) Two or more subsequent `br` elements, (b) a `hypnotic:Separator` element, (c) if the current element belongs to the elements that cause a paragraph-break in the browser (see `@b` in listing 1), (d) if the tree walker leaves an as yet unmarked subtree governed by one of the elements in `@b`, (e) a significant change in font size.

The function `markTextBlocks` takes the array consisting of one or more `hypnotic:Text` elements, under certain conditions interspersed with zero or more `hypnotic:Icon` elements and encapsulates this sequence within `hypnotic:TextBlock`. The insertion of this new element occurs as high as possible in the DOM tree, i. e., we examine the respective parent nodes of the `hypnotic:Text` elements (exploiting the path information added in stage 1) and determine whether there are additional `hypnotic:Text` children which are not part of our current sequence. If this is not the case, we examine the parent of the parent, and so on, until we find the LCN, the least common node to which we can safely attach the `hypnotic:TextBlock` node. Under certain conditions, there is no LCN. In that case, we have to include the new node at the correct position at the child level of one of the common ancestors of the elements in `@s`. We are then able to move all the subtrees that include the elements in `@s` beneath the newly inserted node. Fig. 1 shows a DOM tree modified in this manner.

Stage 4: Processing of Text Blocks

Stage 3 inserts a new level of explicit structure into the document which is only implicitly contained in the source document. Stage 4 further exploits the newly introduced structure by examining the individual subtrees governed by the `hypnotic:TextBlock` nodes in order to detect an even higher level of structure, aggregating one or more `TextBlock` nodes. At the moment, our prototype is able to detect headlines on various levels, 'footnotes' (i. e., one or more subsequent text blocks with a significantly smaller font size than the base font size) and several different types of explicit and implicit list structures.

Detecting explicitly marked lists is straightforward: Using XPath expressions, we locate all the `hypnotic:TextBlock` nodes that govern a `ul`, `ol`, or `dl` node which contains at least one `li` or `dt`/`dd` element, and encapsulate each of the resulting nodes with a separate `hypnotic:List` element. A special case exists for those lists that contain multiple text blocks (e. g., several `li` elements with explicit paragraph boundaries such as `<br><br>`). Detecting implicit lists is a rather complex process carried out by means of another top-

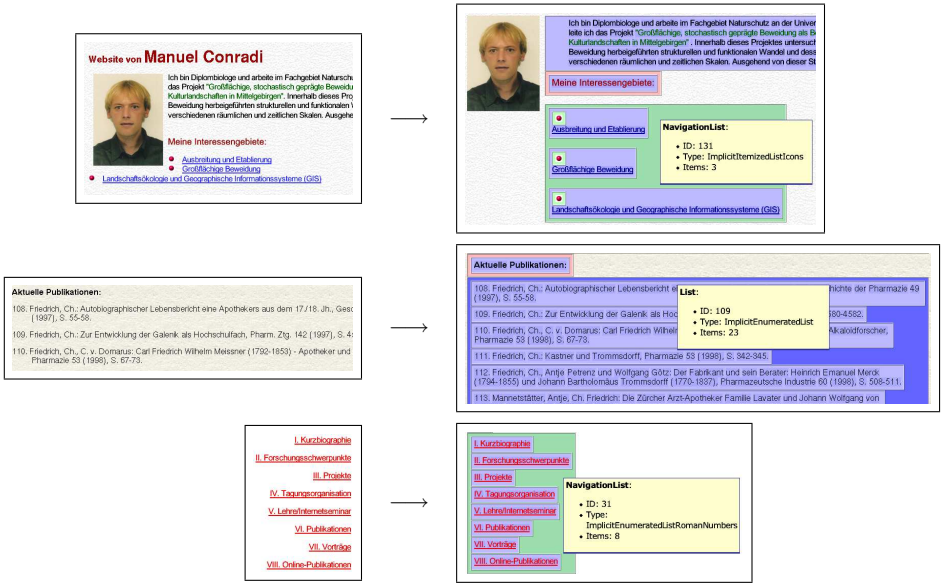

Figure 2: Examples for the detection of different types of implicit lists (see the popup-windows on the right; left: original document as rendered in Mozilla, right: visualization of the analysis in the front end of our corpus-database, see section 4.3).

down/depth-first tree walker which encapsulates certain `hypnotic:TextBlock` sequences as `hypnotic:List`, in some cases interspersed with `hypnotic:Icon`.

Currently, we detect (a) itemized lists in which the bullet point is realized by a small-sized inline image (i. e., sequences of the following structure: `Icon`, 1..*n* `TextBlock`, `Icon`, 1..*n* `TextBlock`, ...), (b) implicit itemized lists in which the bullet point is simulated by typographical means (e. g., a consistent - or * at the beginning of each `TextBlock`), (c) implicit enumerated lists in which the label consists of numbers of increasing values in arabic or (d) roman notation (see fig. 2 for examples). Again, the insertion of the new `List` node uses the algorithm to find the LCN of a given sequence of nodes in order to locate the correct point within the DOM tree (see stage 3). Up to this point, we do not take into account any hyperlinks contained within a `List`.

The headline and footnote detection operates on the font sizes of adjacent text blocks. If these are significantly higher or lower than the base font size, the respective sequence is marked as `Headline` or `Footnote`. Both elements receive an attribute `Level`, marking the relative headline or footnote level (based on the font size). In addition, we detect headlines embedded in text blocks: As the font size within a text block may vary with

a threshold of ± 5, an initial `Text` node with a font size of +5 is marked as an inline `Headline`.

Stage 5: Detecting Different Types of Hyperlink Lists

In the fifth stage several heuristics detect different types of hyperlink lists. For this purpose, the parser determines all instances of `hypnotic:List` that have at least one list item containing a hyperlink. As the information about the number of hyperlinks comprised in a list as well as the different types of hyperlinks are already available, the identification process can be based on the percentage of a hyperlink type: if more than two thirds of all hyperlinks belong to the type `internal`, we assume that these hyperlinks connect nodes of the current hypertext and replace the element `hypnotic:List` by `hypnotic:NavigationList`. Similar replacement rules exist for the remaining types `external` (creates `hypnotic:Hotlist`), `thispage` (`hypnotic:TableOfContents`) and `samedomain` (`hypnotic:Dispenser`, a special type of navigation list; see figure 2 for examples).

Stage 6: Integrating Part-of-Speech-Information

The final stage incorporates a robust syntactic parser. By means of an XPath expression, all `hypnotic:TextBlock` elements are determined. In all of these elements, all `hypnotic:Text` elements are collected. These sequences of characters are used to construct an array whose fields contain whitespace-separated tokens. This array is passed on to the commercial POS tagger and syntactic parser Connexor Machinese Syntax German that returns the analysis result in the form of an XML document. As this tool's tokenisation algorithm is not documented, it is necessary to map the Connexor tokenisation (e. g. "[Paul] [laughs] [.]") to our whitespace-separated token approach ("[Paul] [laughs.]"), because a new `hypnotic:Token` element containing the POS information as an attribute value is constructed for *every* word in the DOM structure.

4.3 Visualization and Examples

The corpus-database and the text parser are accessible by means of a web front end implemented in PHP, which accesses the MySQL database in order to retrieve and display the locally stored documents in a browser (see fig. 4). After activation, the source document is processed by the text parser, and the analysis result, i. e., the augmented and serialized DOM tree, is sent through an XSLT stylesheet which transforms the elements of the `hypnotic:` namespace into tables with specific background colours, so that the results can be conveniently displayed within the front end itself (fig. 2 shows three excerpts of these before/after examples, fig. 4 shows a complete example). Furthermore, the XSLT stylesheet converts the analysis information contained within the attributes of the `Headline`, `List` etc. elements so that they are displayed in a popup-window as soon as the mouse pointer is moved over the respective table region (again, see fig. 2); the required JavaScript code fragments are generated dynamically. For the development

phase of the text parser, this function is a clear advantage, as it is far more user-friendly than examining large amounts of raw, tagged XML markup, e. g., in an XML editor. Furthermore, in some situations, the text parser tends recursively to nest certain elements within each other, e. g., a `TextBlock` within a `TextBlock`. Annotation errors like these can be detected extremely quickly by means of corresponding templates in the XSLT stylesheet: if such an automatic error detection template matches (in this case, `hypnotic:TextBlock//hypnotic:TextBlock`), the stylesheet prints an error message at the beginning of the result document, indicating that something went wrong.

In addition, the web front end provides functions for rendering the source document, the augmented document (see fig. 1), and an HTML-free version of the augmented structure as DOM-like trees (in Postscript or GIF format). These functions are important for verifying that the insertion of `hypnotic:*` nodes works correctly. The reduced version of the augmented structure is realized by means of another XSLT stylesheet that removes all elements of the `xhtml:` namespace (see fig. 3). In order not to obtain a completely flat tree, intermediate nodes (e. g., a `xhtml:p` node that governs two text block nodes) are substituted by a dummy node (`hypnotic:Node`, see the tree on the right hand side in fig. 3).

4.4 Implementation

The text parser is implemented in Perl and uses the modules XML::LibXML (i. e., the Gnome project's libxml2 library providing a DOM Level 2 parser and an XPath engine), XML::LibXSLT (i. e., Gnome's libxslt library). Furthermore, LWP::Simple is used to establish HTTP connections, and Image::Size is utilised to extract the physical dimensions of image files. The automatic generation of DOM trees is based on the GraphViz tree drawing package and a customized version of the Perl module GraphViz::XML.

4.5 Towards Automatic Web Genre Identification

Section 3 describes our theoretical framework and briefly mentions our ultimate goal: devising a web genre-enabled search engine. Our current design for the architecture of this search engine is built on three main components (Rehm, 2005): (a) The text parser is needed to extract as much explicitly- and implicitly-contained structure in a source document as possible. This data, along with certain keywords will comprise the majority of information which is required for, (b) the web genre classifier, which, in our opinion, can be based on one (or more) of the classic machine learning algorithms or approaches (*k*NN, Naive Bayes, C 4.5 etc.). The third component is (c) a set of ontologies/taxonomies, represented in OWL, that specify information to be used in (i) the classification task (e. g., a web genre ontology) and, (ii) the next level of automatic annotation: As soon as the system is able to automatically classify an arbitrary web page of our domain of interest into its respective web genre, we intend to tackle the problem of automatically converting the document into a markup language that explicates the content and structure of the respective web genre. In other words, we want to map individual text blocks (and higher

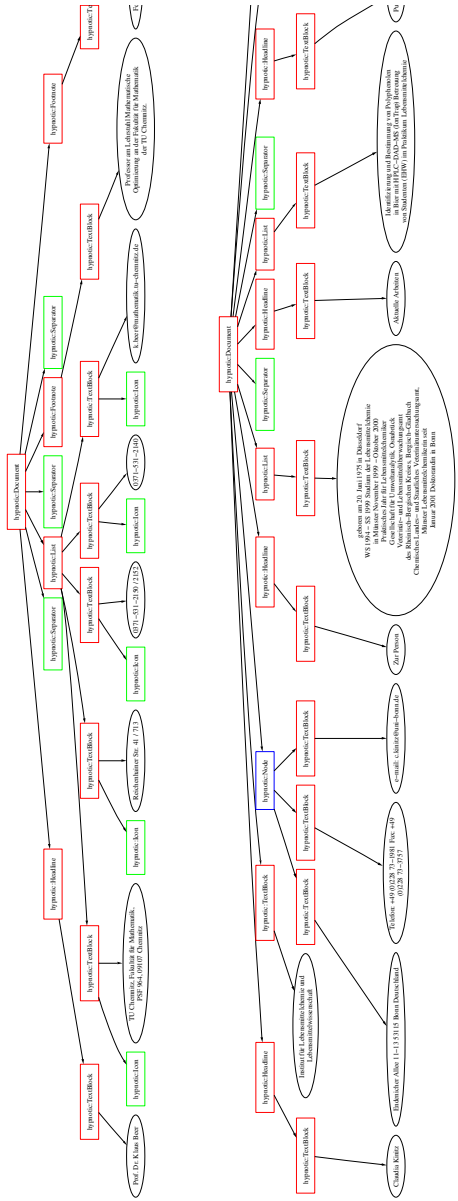

Figure 3: Left: The reduced DOM structure of the document shown in fig. 4. Right: This reduced tree shows that we capture the original HTML-based hierarchical structure by inserting a dummy node (`hypnotic:Node`).

Figure 4: The web front end of the corpus-database in “document view” mode (top: source document, bottom: document as analysed by the text parser and visualized by an XSLT stylesheet; colour key: light blue – text block, dark blue – list, red – separator, light red – headline, green – footnote, light green – icon).

level objects) onto web genre modules. A function like this would, in turn, enable a novel kind of information extraction application; the extraction of the contents of specific web genre modules of web pages that belong to a certain web genre. For these purposes, we envision a comprehensive OWL-/RDF-based format such as the one presented in Potok et al. (2002) to encapsulate the information needed for the mapping algorithm.

5 Conclusions and Future Work

The algorithms of the text parser are quite stable and robust. Of the 100 documents contained in our document sample testbed, only four documents are incorrectly annotated. These documents contain very unusual markup structures which, as yet, cannot be rearranged by the `markTextBlock` algorithm. We will enhance the algorithms as necessary in the near future, so that the complete sample can be analysed correctly. Most documents contained in the sample use `table` elements to realise highly complex layouts. In certain cases, our algorithms are not able correctly to detect the structure contained in these tables. We consciously ignored this problem as we plan to enhance the cascaded processing stages by one additional stage which has to run after the completion of stage 2, in order to (a) distinguish genuine tables from layout-oriented tables, and, to (b) classify each layout-oriented table into the correct form (see Hurst, 2002, Wang et al., 2000, and the other approaches cited in section 2). Furthermore, additional extensions will be implemented to detect other implicit structures and higher level objects. We will add heuristics to examine the contents of text blocks in order to distinguish between text blocks containing only text fragments (such as all the text blocks shown in fig. 4) and those containing continuous text. After mapping each text block into one of these two classes, we can add another layer of processing by identifying individual sentences and annotating the analysis by augmenting the XHTML/XML markup with explicit sentence boundaries.

Apart from the web genre identification application (see section 4.5), our text parser could be used for several other tasks: the hierarchical information which is contained in the individual elements (headline, footnote, text block, list etc.) as attributes, could be used to rearrange the reduced DOM structure in a way that reflects this very hierarchical structuring. A function like this could be used to facilitate browsing on mobile devices (e. g., to display only the headlines first, so that the user can decide which headline to explore further by dynamically folding out the subtree governed by the headline node), or as an initial stage for automatic text summarization approaches, especially in conjunction with the more detailed text block analysis sketched in the previous paragraph. Furthermore, the envisioned automatic sentence boundary detection could be used to gather collections of test sentences for natural language parser research and evaluation (Web as Corpus approach, Kilgarriff, 2001, Rehm, 2003). Another possible application for the text parser is the automatic weighting of certain text fragments within a traditional search engine context: a specific keyword contained in a first-level-headline is definitely more important to a certain web page than the same keyword contained in a 3rd-level-footnote.

References

Alam, H., F. Rahman, Y. Tarnikova, and A. Kumar (2003). When is a List is a List?: Web Page Re-authoring for Small Display Devices. In *Proceedings of WWW 2003*, Budapest.

Barnard, D. T., L. Burnard, S. J. DeRose, D. G. Durand, and C. Sperberg-McQueen (1996). Lessons for the World Wide Web from the Text Encoding Initiative. *The World Wide Web Journal 1*(1), 349–357.

Buyukkokten, O., H. Garcia-Molina, and A. Paepcke (2001). Seeing the Whole in Parts: Text Summarization for Web Browsing on Handheld Devices. In *Proceedings of WWW 2001*, Hong Kong.

Carchiolo, V., A. Longheu, and M. Malgeri (2003). Extracting Logical Schema from the Web. *Applied Intelligence 18*, 341–355.

Chan, M. and G. Yu (1999). Extracting Web Design Knowledge: The Web De-Compiler. In *IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS 1999)*, Volume 2, Florence, pp. 547–552.

Chen, J., B. Zhou, J. Shi, H. Zhang, and Q. Fengwu (2001). Function-Based Object Model Towards Website Adaption. In *Proceedings of WWW-10*, Hong Kong, pp. 587–596.

Chen, Y., W.-Y. Ma, and H.-J. Zhang (2003). Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices. In *Proceedings of WWW 2003*, Budapest.

Chung, C. Y., M. Gertz, and N. Sunaresan (2002). Reverse Engineering for Web Data: From Visual to Semantic Structures. In *Proceedings of the 18th Int. Workshop on Data Engineering (ICDE '02)*, San Jose, pp. 53–63.

Chung, C. Y., M. Gertz, and N. Sundaresan (2001). Quixote: Building XML Repositories from Topic Specific Web Documents. In J. S. Giansalvatore Mecca (Ed.), *Fourth Int. Workshop on the Web and Databases (WebDB 2001)*, Santa Barbara, pp. 103–108.

Clark, J. (1999). XSL Transformations (Version 1.0). Technical Specification, W3C. `http://www.w3.org/TR/xslt/`.

Clark, J. and S. DeRose (1999). XML Path Language (XPath) – Version 1.0. Technical Specification, W3C. `http://www.w3.org/TR/xslt/`.

Cohen, W. W., M. Hurst, and L. S. Jensen (2002). A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. In *Proceedings of WWW 2002*, Honolulu.

Cowie, J. and Y. Wilks (2000). Information Extraction. In R. Dale, H. Moisl, and H. Somers (Eds.), *Handbook of Natural Language Processing*, pp. 241–260. New York, Basel: Marcel Dekker.

Crowston, K. and M. Williams (1997). Reproduced and Emergent Genres of Communication on the World-Wide Web. In *Proc. of the 30th Hawaii Int. Conf. on Systems Sciences (HICSS-30)*, Volume 6, pp. 30–39.

de Saint-Georges, I. (1998). Click Here if You Want to Know Who I Am. Deixis in Personal Homepages. In *Proceedings of the 31st Hawaii Int. Conf. on Systems Sciences (HICSS-31)*, Volume 2, pp. 68–77.

Dillon, A. and B. A. Gushrowski (2000). Genres and the Web: Is the Personal Home Page the First Uniquely Digital Genre? *Journal of the American Society for Information Science 51*(2), 202–205.

Eiron, N. and K. S. McCurley (2003). Untangling Compound Documents on the Web. In *Proceedings of the 14th ACM Conf. on Hypertext and Hypermedia*, pp. 85–94. Nottingham.

Fielding, R. T., J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, and P. J. L. T. Berners-Lee (1999). Hypertext Transfer Protocol – HTTP/1.1. Network Working Group – Request for Comments (RFC). Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach und Tim Berners-Lee. `http://www.ietf.org/rfc/`.

Furuta, R. and C. C. Marshall (1996). Genre as Reflection of Technology in the World-Wide Web. In S. Fraïssé, F. Garzotto, T. Isakowitz, J. Nanard, and M. Nanard (Eds.), *Hypermedia Design, Proc. of the Int. Workshop on Hypermedia Design (IWHD 1995)*, pp. 182–195. Berlin, Heidelberg, New York etc.: Springer.

Gupta, S., G. Kaiser, D. Neistadt, and P. Grimm (2003). DOM-based Content Extraction of HTML Documents. In *Proceedings of WWW 2003*, Budapest.

Haas, S. W. and E. S. Grams (2000). Readers, Authors, and Page Structure – A Discussion of Four Questions Arising from a Content Analysis of Web Pages. *Journal of the American Society for Information Science 51*(2), 181–192.

Haase, M., M. Huber, A. Krumeich, and G. Rehm (1997). Internetkommunikation und Sprachwandel. In R. Weingarten (Ed.), *Sprachwandel durch Computer*, pp. 51–85. Opladen: Westdeutscher Verlag.

Hors, A. L., P. L. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne (2000). Document Object Model (DOM) Level 2 Core Specification. Technical Specification, W3C.

Hurst, M. (2002). Classifying TABLE Elements in HTML. In *Proceedings of WWW 2002*, Honolulu.

Kilgarriff, A. (2001). Web as Corpus. In P. Rayson, A. Wilson, T. McEnery, A. Hardie, and S. Khoja (Eds.), *Proceedings of the Corpus Linguistics 2001 Conf.*, Lancaster, pp. 342–344.

Lim, S.-J. and Y.-K. Ng (1999). An Automated Approach for Retrieving Hierarchical Data from HTML Tables. In *Proceedings of the 8th Int. Conf. on Information and Knowledge Management (CIKM '99)*, pp. 466–474. ACM Press.

Mehler, A., M. Dehmer, and R. Gleim (2004). Towards Logical Hypertext Structure – A Graph-Theoretic Perspective. In T. Böhme and G. Heyer (Eds.), *Proceedings of the Fourth Int. Workshop on Innovative Internet Computing Systems (I2CS '04)*, Lecture Notes in Computer Science, Berlin, New York. Springer.

Myllymaki, J. (2001). Effective Web Data Extraction with Standard XML Technologies. In *Proceedings of WWW-10*, Hong Kong, pp. 689–696.

Penn, G., J. Hu, H. Luo, and R. McDonald (2001). Flexible Web Document Analysis for Delivery to Narrow-Bandwidth Devices. In *Int. Conf. on Document Analysis and Recognition (ICDAR '01)*, Seattle, pp. 1074–1078.

Potok, T. E., M. T. Elmore, J. W. Reed, and N. F. Samatova (2002). An Ontology-based HTML to XML Conversion Using Intelligent Agents. In *Proceedings of the 35th Hawaii Int. Conf. on System Sciences (HICSS-35)*, Big Island, Hawaii.

Raggett, D., A. L. Hors, and I. Jacbos (1999). HTML 4.01 Specification. Technical Specification, W3C. `http://www.w3.org/TR/html401/`.

Rehm, G. (2001). `korpus.html` – Zur Sammlung, Datenbank-basierten Erfassung, Annotation und Auswertung von HTML-Dokumenten. In H. Lobin (Ed.), *Proceedings of the GLDV Spring Meeting 2001*, Giessen, Germany, pp. 93–103. Gesellschaft für linguistische Datenverarbeitung. `http://www.uni-giessen.de/fbo9/ascl/gldv2001/`.

Rehm, G. (2002). Towards Automatic Web Genre Identification – A Corpus-Based Approach in the Domain of Academia by Example of the Academic's Personal Homepage. In *Proceedings of the 35th Hawaii Int. Conf. on System Sciences (HICSS-35)*, Big Island, Hawaii.

Rehm, G. (2003). Texttechnologie und das World Wide Web – Anwendungen und Perspektiven. In H. Lobin and L. Lemnitzer (Eds.), *Texttechnologie – Anwendungen und Perspektiven*, Stauffenburg Handbücher, pp. 433–464. Tübingen: Stauffenburg.

Rehm, G. (2004a). Hypertextsorten-Klassifikation als Grundlage generischer Informationsextraktion. In A. Mehler and H. Lobin (Eds.), *Automatische Textanalyse – Systeme und Methoden zur Annotation und Analyse natürlichsprachlicher Texte*, pp. 219–233. Wiesbaden: Verlag für Sozialwissenschaften.

Rehm, G. (2004b). Ontologie-basierte Hypertextsorten-Klassifikation. In A. Mehler and H. Lobin (Eds.), *Automatische Textanalyse – Systeme und Methoden zur Annotation und Analyse natürlichsprachlicher Texte*, pp. 121–137. Wiesbaden: Verlag für Sozialwissenschaften.

Rehm, G. (2005). *Hypertextsorten: Definition – Struktur – Klassifikation*. Ph. D. thesis, Institut für Germanistik, Angewandte Sprachwissenschaft und Computerlinguistik, Justus-Liebig-Universität Gieß en.

Shepherd, M. and C. Watters (1999). The Functionality Attribute of Cybergenres. In *Proceedings of the 32nd Hawaii Int. Conf. on Systems Sciences (HICSS-32)*.

Song, R., H. Liu, J.-R. Wen, and W.-Y. Ma (2004). Learning Block Importance Models for Web Pages. In *Proceedings of WWW-2004*, New York, pp. 203–211. ACM Press. Refereed Papers Track.

Wang, H.-L., S.-H. Wu, I. C. Wang, C.-L. Sung, W. L. Hsu, and W. K. Shih (2000). Semantic Search on Internet Tabular Information Extraction for Answering Queries. In *Proceedings of the 9th Int. Conf. on Information and Knowledge Management (CIKM 2000)*, McLean, pp. 243–249. ACM Press.

Wang, Y. and J. Hu (2002). A Machine Learning Based Approach for Table Detection on The Web. In *Proceedings of WWW 2002*, Honolulu.

Yang, G., S. Mukherjee, W. Tan, I. V. Ramakrishnan, and H. Davulcu (2003). On the Power of Semantic Partitioning of Web Documents. In S. Kambhampati and C. A. Knoblock (Eds.), *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, pp. 21–26.