Ontology-Based HTML to XML Conversion

Shijun Li[1,2], Weijie Ou[1], and Junqing Yu[3]

[1] School of Computer, Wuhan University, Wuhan 430072, China
shjli@public.wh.hb.cn, ouwj1981@sohu.com
[2] Laboratory of Computer Science, Institute of Software,
Chinese Academy of Science, Beijing 100080, China
[3] College of Computer Science & Technology,
Huazhong University of Science & Technology, Wuhan 430074, China
yjqinghust@126.com

Abstract. Current wrapper approaches break down in extracting data from differently structured and frequently changing Web pages. To tackle this challenge, this paper defines domain-specific ontology, captures the semantic hierarchy in Web pages automatically by exploiting both structural information and common formatting information, and recognizes and extracts data by using ontology-based semantic matching without relying on page-specific formatting. It is adaptive to differently structured and frequently changing Web pages for a domain of interest.

1 Introduction

Efficiently extracting data from Web pages and converting them into XML have many potential applications [6]. Unfortunately, since most Web pages are in HTML, which is not designed to describe the content of data, there is no simple way to capture the semantic content in Web pages. Current wrapper approaches work well in extracting data from Web sources with the same structure, or with large sets of pages generated using a common temple by searchable databases online [1,10], but break down in extracting data from Web sources with differently structured and frequently changing pages for a domain of interest.

In this paper, we study the problem of extracting data from the differently structured and frequently changing pages for a domain of interest. As XML describes the contents of the data and can be queried and processed by a database application [9], we use XML to represent the captured data. We define the domain-specific ontology as the shared conceptualization of knowledge in a particular domain, which consists of thesaurus, label, keywords, begin-landmark, end-landmark, match and separator. According to HTML grammar, an HTML document consists of a sequence of *blocks* that may be *heading*, *paragraph*, *list*, and *table* etc. , which may not be in the same semantic hierarchy. As a *heading* element briefly describes the topic and semantic hierarchy of the section it introduces, we can capture the inter-block semantic hierarchy of an HTML document according to different levels of *heading* in it. In such cases of using common formatting information, rather than using *heading* to show the semantic hierarchy, we can capture the inter-block semantic hierarchy by the common formatting information. After capturing the inter-block semantic hierarchy, we exploit the structural information that HTML tags imply, such as a *list* consists of *items*, to capture the intra-block semantic hierarchy. After the semantic hierarchy structures in Web pages be captured, then we can recognize and extract concepts by using ontology-based semantic matching.

The rest of the paper is organized as follows. Section 2 defines domain-specific ontology. Section 3 and Section 4 present the approach to capturing inter-block and intra-block semantic hierarchy in Web pages, respectively. Section 5 discusses related work. Finally we conclude in Section 6.

2 Domain-Specific Ontology

Domain-specific ontology is the shared conceptualization of knowledge in a particular domain. It consists of concepts and relations among the concepts. We formally define domain ontology as follows.

Definition 1. *Domain ontology* O_D *is denoted by* $O_D(C, R)$*, where* C *is the set of concepts in domain* D*,* R *the set of relation among the concepts in* C*,* $R = \{(c_1, c_2, r) \mid c_1 \in C, c_2 \in C$*, where* r *is the relation name of* c_1 *and* $c_2\}$.

A Web page is a string of tokens. We define a token to be a word, an HTML token or a punctuation, and a string to be a string of tokens. As different Web page may use different words to express the same concept in a domain, we need thesaurus of a concept and global name for each concept in a domain. And in order to recognize a concept in a Web page, we use *thesaurus*, *label*, *keywords*, *begin-landmark*, *end-landmark*, *match* and *separator*, which are page-specific independent and are formally defined below, to relate a concept.

Definition 2. *A concept* c *in a domain is denoted by:* $c =$ *(name, thesaurus, label, keywords, begin-landmark, end-landmark, match, separator), where name is the global name of the concept; thesaurus is the set of all the synonym corresponding to the concept; label and keyword are set of strings to mark a nested concept name and the content of a concept, respectively; begin-landmark and end-landmark are the set of strings that mark the begin and end of the concept, respectively, where we use* b*,* p*,* t *to denote blank, punctuation, and HTML tags; match is a regular expression or a file name used to match the concept; separator is set of strings to separate the instances, if the concept has multiple instances. All the attributes except* name *are optional.*

In this paper, we use XML DTD to represent domain ontologies. A concept c is called a primary key concept, if it can distinguish different instances of the root concept in a domain ontology. For example, in the domain ontology of home page, *name* is the primary key concept (suppose there is no duplicate name). And for the concept of ResearchInterest, thesaurus = {research interest; domain of research; recent research}, label = {research interest; domain of research; recent research}, end-landmark = {b, t}, separator = {",", ";"}. And a string is called a simple string if it contains no labels and keywords of any concept in a domain ontology that marks nested concepts. To capture the nested concepts, we introduce three operators related to a domain ontology as follows.

Definition 3. *Let s be a string, the domain ontology O_D. Then, $\gamma_D(s) = c$, if $\exists c \in C$ and $\exists s\prime$, where $s\prime$ is a sub-string of s, and $s\prime \in c$.thesaurus (c.thesaurus denotes the thesaurus attribute of c); otherwise, $\gamma_D(s) = s$. If there is a substring $s\prime$ in s that match the c.match of a concept c, then $\mu_c(s) = s\prime$. Let l be a label or key of a concept c in the domain ontology. Then $\alpha_D(l) = c$.*

For example, let s = recent publications, and *publishing* is a concept name in the domain of home pages, then $\gamma_D(s)$ = publishing; let l = recent areas of research, then $\alpha_D(l)$ = ResearchInterest; let s = Tom Bush's home page, then $\mu_{name}(s)$ = Tom Bush. In our system, a domain ontology may be provided manually by system users, domain experts, knowledge engineers, or automatically generated based on induction. It is generated once, but can be used many times.

3 Capturing Inter-block Semantic Hierarchy

3.1 Exploiting Structural Information

An HTML document consists of a *head* element and a *body* element, and must have a *title* element in the head section. We capture the matched content of title as the content of the key concept in the domain ontology as follows.

Let D be an HTML document as follows: D =<html><head> <title>T</title>... <body>B</body></html>. Then $\psi(D) =< R >< K > \mu_K(T) < /K > \psi(B) < /R >$, where R, K are the name of the root element and the name of the key concept in the domain ontology, respectively, and ψ the converting function that converts an HTML document into an XML one.

The *body* element consists of a sequence of *blocks* that may be *heading*, *paragraph*, *table*, and *list* etc., but the sequence of blocks that are partitioned according to HTML grammar may not be in the same semantic hierarchy. As most HTML documents use *heading* blocks to express its semantic hierarchy, whose six levels from the most important (*h1*) to the least important (*h6*) briefly describe the topic and semantic hierarchy of the block it introduces, we can use it to capture the semantic hierarchy of the sequence of blocks by converting the content of each of the most important *heading* into an XML element, and convert the block that the *heading* introduces as its content as follows, recursively.

Let D be an HTML section: $D = S_0$<hn>T_1</hn>$S_1 \ldots$<hn>T_m</hn >S_m, where hn is the most important heading in D, and each of $S_0, \ldots, S_m$ is a sequence of HTML blocks. *Then $\psi(D) = \psi(S_0) \oplus \psi(S_1^{'}) \oplus \ldots \psi(S_i^{'}) \oplus \ldots \psi(S_m^{'})$, where $\oplus$ denotes concatenate operation of two strings, $S_i^{'}$ =<hn>T_i</hn >S_i, and $i = 1, ..., m$. If T_i is a simple string without link then $\psi(S_i^{'})$ =<$\gamma(T_i)$>$\psi(S_i)$ </$\gamma(T_i)$>; If T_i is a simple string but with a link point to a page, suppose the body of this page is B, then $\psi(S_i^{'})$ =<$\gamma(T_i)$>$\psi($<B>$) \oplus \psi(S_i)$</$\gamma(T_i)$>; If T_i is not a simple string, that is, it contains nested concepts recognized by labels or keywords, then $\psi(S_i^{'}) = \psi(T_i) \oplus \psi(S_i)$.*

The above converting function is recursive, we can recursively apply it till the whole semantic hierarchy in a Web page is captured.

3.2 Exploiting Formatting Information

Some Web pages may use formatting information, rather than using any *heading* blocks to show their semantic hierarchy. The formatting information that can be exploited for capturing semantic hierarchy includes *bold* tags, *strong* tags and the biggest font etc. In such case, the key is to find out all the text strings that have the most important formatting information, whose function is just as the most important *heading* in an HTML document. Take it as the important heading elements, then we can use the above approach to capture the semantic hierarchy and convert it into XML. Note that we can recursively use the strongest formatting information and take it as the heading elements, so that we can capture the whole semantic hierarchy in a Web page.

4 Capturing Intra-block Semantic Hierarchy

4.1 Paragraphs

After recursively capturing the inter-block semantic hierarchy, the next step is capturing the intra-block semantic hierarchy. First, consider the *p* element. If the content of a *p* element is a simple string, we need do nothing. Otherwise, the content of a *p* element contains nested concepts marked by labels and keywords in domain ontology. We capture the nested concept of *p* element as follows.

Let $D =< p > S < /p >$*. If* S *is a simple string, then* $\psi(D) = s$*. Otherwise; if* S *contains nested concepts, let* $S = S_1CS_2$*, if* C *is a keyword related to a concept* c*, then* $\psi(D) = \psi(S_1)\oplus < c > C < /c > \oplus\ \psi(S_2)$*; if* $C = C_1TC_2$*, where* C_1 *and* C_2 *are the label and the separate related a concept* c *in domain ontology, respectively, then* $\psi(D) = \psi(S_1)\oplus < c > \psi(T) < /c > \oplus\ \psi(S_2)$*.*

4.2 Lists

In HTML documents, both unordered lists *ul* and ordered lists *ol* consist of *li* elements, which are their items. All the *li* elements logically belong to the same level, and can be taken as the children elements of its parent concept. So we can convert all the *li* elements into the children concept of the parent concept in the domain ontology as follows.

Let $L =< ul > L_1 \ldots L_n < /ul >$*, or* $L =< ol > L_1 \ldots L_n < /ol >$*, where* L_i *is a* li *element and* $i = 1, \ldots, n$*. Let the name of the concept recognized before* L *is* b*. If* b *has a children elements* c *in domain ontology DTD, then* $\psi(L) = \psi(L_1) \oplus \ldots \oplus \psi(L_n) =< c > \psi(s_1) < /c > ... < c > \psi(s_n) < /c >$*; Otherwise,* $\psi(L) = \psi(L_1) \oplus \ldots \oplus \psi(L_n) =< b > \psi(s_1) < /b > ... < b > \psi(s_n) < /b >$*, where* $s_1, ..., s_n$ *is the content of* $L_1, ..., L_n$*, respectively.*

4.3 Tables

In Web pages, table cells generally contain heading information via the *th* element and data via the *td* element, and may span multiple rows and columns. If a *th* or *td* element contains $colspan = n$ or $rowspan = n$, the particular cell of the *th* or *td* is to be expanded to $n - 1$ more columns, starting from the current cell in the current row, or to the next $n - 1$ rows in the current column, respectively. By inserting redundant cells according to the attributes of *colspan* and *rowspan*, we can normalize an HTML table, in which each row has the same number of cells aligned. The attribute-value pairs in the normalized table is what we want to captured. The mapping rule is suited for the HTML tables with marked headings via *th* elements. To convert HTML tables without marked headings via *th* elements, we can recognize their headings by formatting information, which are mainly font (size, bold) [4]. In the case of tables are for creating some type of multiple-column layout for easy viewing, we just ignore the table tags.

4.4 Frames

HTML frames allow authors to present documents in multiple views. An HTML document that describes frame layout (called a frameset document) has a HEAD, and a FRAMESET in place of the BODY. To capture the semantic hierarchy in a frame Web pages equals to capture each frame in the page.

Let D =<frameset...> <frame="frame1.html">... <frame="framen.html"> </frameset>, and f_1, ..., f_n *denote frame1.html, ..., framen.html, respectively. Then* $\psi(D) = \psi(f_1) \oplus ... \oplus \psi(f_n)$.

5 Related Work

The manually or semiautomatic approaches [8], and the induction approaches [3] to extract wanted data from a Web site work well in extracting data from multiple Web pages with the same structure, but break down when web pages changed and for large scale of differently structured web pages. The ontology-based or concept schema-based approaches [2,7,5] create a domain-specific wrapper called an extraction ontology, rather than creating a page-specific wrapper. These approaches work well in the regularly formatted pages such of news papers Web sites, but breaks down in the domain where the page formatting can be more complex such as the domain of home pages. These approaches also partly rely on the page-specific formatting. In this paper, we improve our earlier work [4] by using ontology. A system based on the approach presented here has been implemented. We have performed experiment to extract data on the home pages of researchers in universities with which existed related approaches have poor performance. The domain ontology of home pages is generated easily by manually. The average precision (number of correctly extracted concepts / number of all extracted concepts) and the recall (number of correctly extracted concepts / number of all the concepts that should be correctly extracted) of our approach are approximately 81%, and 74%, respectively, which are calculated by manually based on experiment data on randomly chose 100 home pages. Compared with other related approaches, our approach has the following advantages.

(1)It does not rely on any page-specific formatting to guide extraction, and need no any training on the format of those pages. It is adaptive for loosely structured pages and resilient to changes in pages for a domain of interest.

(2) It exploits both formatting and structural information implied by HTML tags to automatically capture the semantic hierarchy, and finer granularity information in Web pages based on domain ontology. Moreover, it uses recursive function, so it can handle arbitrarily nested structure in Web pages.

6 Conclusion

This paper defines the domain-specific ontology, exploits both structural information and common formatting information to automatically capture the nested semantic hierarchy, and recognize and extract data in Web pages by using ontology-based semantic matching. It is adaptive with loosely structured pages and resilient to changes in pages for a domain of interest.

References

1. Arvind Arasu and Hector Garcia-Molina. Extracting Structured Data from Web Pages. In Proc. VLDB (2003) 337-348
2. David W. Embley, Cui Tao, and Stephen W. Liddle: Automatically Extracting Ontologically Specified Data from HTML Table of Unknown Structure. In ER 2002 (2002) 322–337
3. Nicholas Kushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In IJCAI (1997), 729–737
4. Shijun Li, Mengchi Liu, Tok Wang Ling, and Zhiyong Peng. Automatic HTML to XML Conversion. In Proc. WAIM (2004), 714–719
5. Xiaofeng Meng, Hongjun Lu, Haiyan Wang, and Mingzhe Gu. Data Extraction from the Web Based on Pre-Defined Schema. J. Comput. Sci. Technol, 17(4)(2002):377–388
6. Zhiyong Peng, Qing Li, Ling Feng, Xuhui Li, and Junqiang Liu. Using Object Deputy Model to Prepare Data for Data Warehousing. IEEE Transaction on Knowledge and Data Engneering, Vol. 17, NO. 9, (2005)
7. Thomas E. Potok, Mark T. Elmore, Joel W. Reed, and Nagiza F. Samatova. An Ontology- Based HTML to XML Conversion Using Intelligent Agents. In J.Loyd et al., editor, HICSS (2002), 120–129.
8. Arnaud Sahuguet and Fabien Azavant. Building Intelligent Web Applications Using Lightweight Wrappers. Data and Knowledge Engineering (2001), 36(3): 283–316
9. Guoren Wang, Bing Sun, Jian-Hua Lv, and Ge Yu. RPE Query Processing and Optimization Techniques for XML Databases. J. Comput. Sci. Technol, 19(2)(2004):224–237
10. Wensheng Wu, Clement T. Yu, AnHai Doan, and Weiyi Meng. An Interactive Clustering-baded Approach to Integrating Source Query interfaces on the Deep Web. In Proc. SIGMOD (2004), 95–106